PROYECTO FINAL DE CARRERA

APLICACIÓN WEB DE CÁLCULO DE ESTRUCTURAS

(WEB APP FOR STRUCTURE CALCULUS)

Estudios: Ingeniería de Telecomunicación

Autor: Carlos de la Fuente Antequera

Director: Jordi Forga Alberich

Año: 2016

Índice general

Colaboraciones

Colaboración con el Departamento de Ingeniería de la Construcción de la Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos de la Barcelona.

Agradecimientos

Principalmente, quiero agradecer a mi familia por la paciencia que han tenido conmigo durante la realización del proyecto y, en general, durante toda la carrera. Gracias, en concreto a mis padres, por haberme pagado los estudios y por apoyarme en todos los momentos difíciles, como padres y como amigos.

Gracias a mi director de proyecto, actualmente profesor de la escuela, Jordir Forga Alberich; por la ayuda que me ha ofrecido en todo momento y por la predisposición a aclarar las dudas surgidas con tan buena voluntad.

Gracias a Xavier Muñoz, profesor de la escuela y amigo, por su apoyo en muchos de los momentos de dificultad en los que me he encontrado; sin su ayuda no hubiese podido superarlos.

Gracias a Dios por haberme dado otra oportunidad en la vida: “La aprovecharé, te lo aseguro”.

Resum del Projecte

Amb aquest projecte es du a terme el desenvolupament d'un servei web per realitzar càlculs d'estructures de formigó de secció rectangular. Es parteix de la base que existeix un programa fet amb llenguatge de càlcul tècnic específic per al càlcul d'estructures, concretament el programa està implementat amb llenguatge de Matlab.

Tot i que l'objectiu és molt concret; donat que és un servei web, han sorgit molts altres petits objectius propis d'un servei web de qualitat; com són: el control de diversos atacs informàtics, gestió d'usuaris, gestió de variables pròpies de l'aplicació com són els diversos estàndards de formigó y acer (variants al llarg del temps), entre d'altres.

Encara que el Matlab no pertany al repositori de programari lliure, gràcies al component d'execució de Matlab (MCR, de les sigles en anglès Matlab Component Runtime) no és necessari tenir el programa Matlab instal·lat al servidor i, per tant, no són necessàries cap de les seves llicències per distribuir el servei a internet; tot i que es disposi de les seves llibreries de càlcul.

En resum, el projecte consisteix en: implementar les funcionalitats pròpies d'un servei web amb gestió d'usuaris, generar la interfície gràfica que permeti la inserció de variables d'entrada, mostrar els resultats que generi el procés de Matalb i, finalment, la comunicació entre el servidor web i el procés del MCR.

Resumen del Proyecto

Con este proyecto se ha desarrollado un servicio web para realizar cálculos de estructuras de hormigón de sección rectangular. Se parte de la base que existe un programa desarrollado con lenguaje de cálculo técnico específico para el cálculo de estructuras, concretamente el programa está implementado con lenguaje Matlab.

Aunque el objetivo es muy concreto; dado que es un servicio web, han surgido otros objetivos propios de un servicio web de calidad; como son: el control de algunos ataques informáticos, gestión de usuarios, gestión de variables propias de la aplicación tales como diversos estándares de hormigón y acero (variantes a lo largo del tiempo), entre otras.

Si bien el Matlab no es un programa de licencia libre, gracias al componente de ejecución de Matlab (MCR, de las siglas en ingles Matlab Component Runtime) no es necesario tener el programa Matlab instalado en el servidor y, por lo tanto, no es necesaria ninguna de sus licencias para distribuir el servicio por internet; aunque dispongamos de sus librerías de cálculo.

En resumen, el proyecto consiste en: implementar las funcionalidades propias de un servicio web con gestión de usuarios, generar la interface gráfica que permita la inserción de variables de entrada i mostrar los resultados que genere el proceso de Matlab (ejecutado dentro del MCR) i, finalmente, la comunicación entre el servidor web i el proceso del MCR.

Abstract

This project is in charge of creating a web application service for rectangular-section concrete structure calculus. It is a web interface using an existing Matlab program which has all the necessary technical programming code statements to do the calculus.

Although the target of the project is very specific, other web application questions arise from the fact of creating a quality service. Some of them were: basic hacker attack security concerns, user managing, some selectable values managing for the structure calculus such as concrete types or steel types, among others.

While Matlab is a proprietary product of MathWorks, it offers a plugin tool for executing Matlab applications without having to install Matlab at all but only the plugging MCR (Matlab Component Runtime), which can be downloaded from http://es.mathworks.com/products/compiler/mcr/ for the proper version of Matlab. No license is needed to execute Matlab applications inside MCR but there is yet the need of Matlab for developing.

In summary, the project consists on: developing basic web app functionality such as user managing, creating input and output user interface for the structure calculus by the Matlab app inside the MCR and, finally, the inner communication between the web app and the Matlab app.

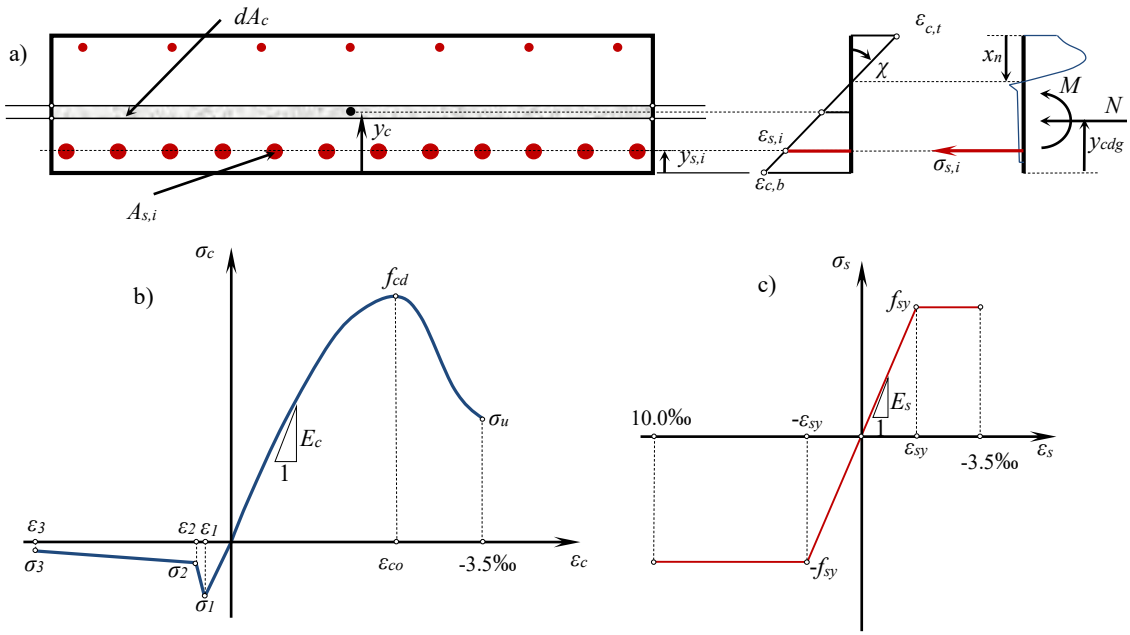
1. Introducción

En este primer capítulo de introducción se pone en contexto el proyecto para ubicar al lector de dónde se parte, se detallan los objetivos del proyecto a los cuales se pretende llegar y, finalmente, hay un pequeño resumen de la estructura que se ha seguido para llegar a ellos.

1.1 Contexto del proyecto

Este proyecto se basa en una rutina de Matlab realizada por el Doctor Ingeniero de Caminos: Albert de la Fuente Antequera. Dicha rutina implementa un método de cálculo llamado Análisis Estructural de Secciones, que se comenta, a modo teórico, a continuación:

El modelo AES (Análisis Estructural de Secciones) es un código implementado en lenguaje MATLAB para el análisis mecánico de secciones con geometría rectangular (ver la siguiente ilustración, 1.1).



$\sigma_1 = f_{ct}$ y $\varepsilon_1 = \sigma_1 / E_c$
$\sigma_2 = f_{Fts} = 0.45 f_{R,1}$ y $\varepsilon_2 = \varepsilon_{SLS} = \mathrm{CMOD}_1 / l_{cs}$
$\sigma_3 = f_{Ftu} = f_{Fts} - w_u (f_{Fts} - 0.5 f_{R,3} + 0.2 f_{R,1}) / \mathrm{CMOD}_3 \geq 0$ y $\varepsilon_3 = \varepsilon_{ULS} = \min(\varepsilon_{Fu}, 2.5 / l_{cs})$

Ilustración 1-1: (a) Sección rectangular de hormigón reforzado con fibras y armadura pasiva; (b) diagrama constitutivo del hormigón para compresión (ε < 0) y para tracción en caso de hormigón reforzado con fibras (ε ≥ 0) y (c) diagrama constitutivo bi-lineal para la armadura pasiva.

Las secciones rectangulares son las más habituales para vigas y pilares de edificación. En este sentido, el modelo permite simular la respuesta mecánica de secciones de hormigón reforzado con fibras y/o con armadura pasiva en forma de redondos. En particular, en el modelo se ha implementado una subrutina que permite obtener el campo tenso - deformacional de los materiales que componen la sección (**σ**, **ε**) cuando ésta está sujeta a unos esfuerzos (**N**,**M**).

1.2 Objetivos

Con este proyecto se trata de desarrollar una aplicación web que permita calcular estructuras de hormigón con sección rectangular. Ello conlleva a tener que presentar una interfaz web con parámetros básicos de la estructura; unos datos de entrada proporcionados por el usuario y los datos de salida correspondientes – calculados mediante un programa externo hecho en Matlab.

La aplicación web debe ofrecer los siguientes servicios:

- Permitir calcular los parámetros básicos de una estructura de sección rectangular.
- Mostrar los parámetros de la estructura calculada en formato web y/o en formato PDF.
- Dado que va a haber parámetros de entrada fijos, para el cálculo de la estructura, éstos deben poder gestionarse.
- La aplicación debe permitir gestionar, también, sus usuarios.

1.3 Estructura de la memoria

La memoria está enfocada a dar solución a la propuesta de proyecto hecha desde el Departamento de Ingeniería de la Construcción de la Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos de Barcelona.

Se estructura tal como se enseñó en la asignatura PAST (Programación Avanzada para Sistemas de Telecomunicación) enfocada a presentar los conceptos básicos del proceso de desarrollo de programas orientados a objetos. Inicia un capítulo de conocimientos previos y, respecto a unos requerimientos específicos para la aplicación, se desarrollan unos casos de uso que debe cumplir, sobre los cuales se basa el diseño; con la descripción del sistema y sus componentes, así como el diagrama de secuencia del principal caso de uso. Finalizan los apartados de conclusión, apéndice y referencias (bibliografía).

2. Conocimientos previos

En este capítulo se detallan todos los conocimientos necesarios para el desarrollo y seguimiento del proyecto, tanto a nivel de lenguajes como de herramientas usados. Asimismo, se ha intentado estructurar el capítulo de manera que los conocimientos más básicos sean los primeros en ser comentados para que no aparezcan conceptos nuevos antes de su previa explicación, necesaria para su puesta en contexto.

Dado que la mayoría de los conceptos que se usan son muy extensos – ya han dado para escribir multitud de libros – en este capítulo se desarrollan muy brevemente cada uno de ellos, y se va directamente al grano con las características/conceptos utilizados. Al mismo tiempo, para la completa descripción y explicación de los mismos, se incluyen referencias y bibliografía consultadas.

2.1 Protocolo HTTP

El protocolo HTTP (Protocolo de transferencia de hipertexto o *Hypertext Transfer Protocol*), es el estándar de comunicación web entre cliente y servidor desarrollado inicialmente en 1991 por el consorcio WWW, en inglés: *World Wide Web Consortium* (W3C), y el Grupo de Trabajo de Ingeniería de Internet, en inglés: *Internet Engineering Taks Force* (IETF). La última versión del protocolo HTTP/2, cuyo RFC (*Request For Comments*) es el RFC 7540, data del mes de mayo del 2015 y es compatible con la penúltima versión HTTP/1.1 de junio del 1999 en cuanto a sintaxis se refiere.

Es un protocolo de nivel de aplicación y está orientado a transacciones de petición/respuesta por parte de cliente/servidor; se basa en mensajes de tipo texto entre cliente y servidor, de manera que: para cada mensaje enviado por el cliente en calidad de petición, recibe una respuesta por parte del servidor.

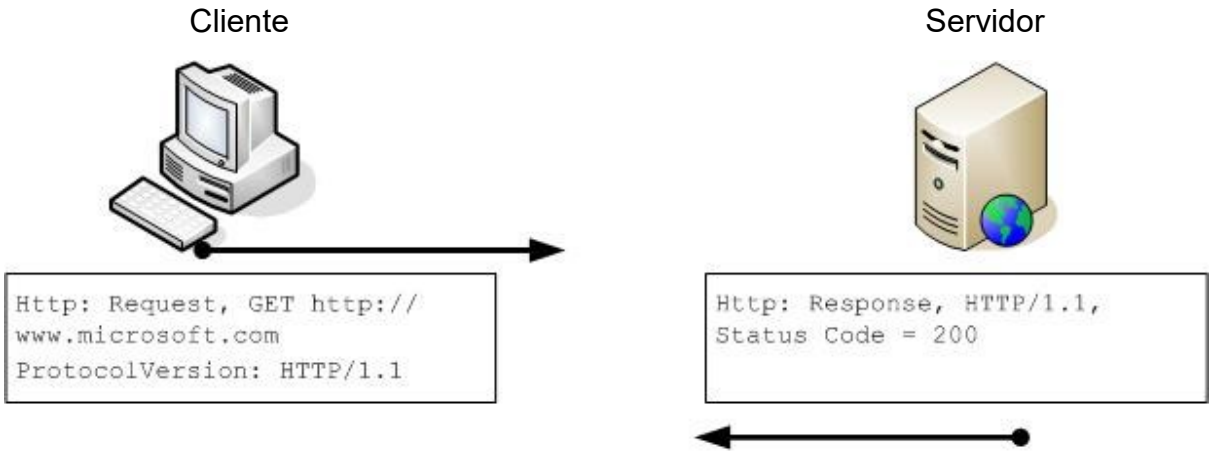


Ilustración 2-1

El cliente envía una petición al servidor con un método de petición o "verbo", una URI (*Uniform Resource Identifiers*) y la versión del protocolo que soporta; seguida por un mensaje de tipo MIME (*Multipurpose Internet Mail Extensions*): Contiene un verbo o modificador de petición, información de cliente y cuerpo de mensaje, y todo sobre una conexión con el servidor.

El servidor responde con una línea de estado; incluye la versión del protocolo que usa para el mensaje de respuesta, seguida de un código de error o de éxito y un mensaje tipo MIME que contiene información sobre el servidor, meta-información de entidad y el opcional cuerpo con el contenido del mensaje.

Algunos de los verbos/modificadores de petición permitidos, más usadas, por este protocolo son los siguientes:

GET

RFC 2616. Pide una representación de la entidad especificada por la URI. Por seguridad no debería ser usado por aplicaciones que causen efectos, ya que transmite información a través de la URI agregando parámetros a la URL. Además, algunos servidores, *proxies* o *user agents* pueden crear *logs* con las peticiones; implícitamente guardando los parámetros que se envíen por este tipo de peticiones y, dando a conocer a terceros información confidencial y sensible de manera indeseable.

Ejemplo: GET /images/boton.png HTTP/1.1 - obtiene un recurso llamado boton.png

Ejemplo con parámetros: GET /index.php?page=main&lang=es

Ref: https://tools.ietf.org/html/rfc2616#page-53

POST

RFC 2616. Envía los datos para que sean procesados por el recurso identificado. Los datos se incluyen en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas. Respuestas posibles de éxito a la petición son las siguientes:

- 200 (OK) o 204 (sin contenido): La acción llevada a cabo por la petición POST no ha generado un recurso que se pueda identificar por una URI. Con la primera (200) la respuesta contiene una entidad que describe el resultado de la petición.
- 201 (creado): Un recurso ha sido creado y la respuesta contiene una entidad con el estado de la petición, referencia al nuevo recurso y una cabecera de localización.

Ejemplo: POST http://localhost:8080/users HTTP/1.1

Ref: https://tools.ietf.org/html/rfc2616#page-54

PUT

RFC 2616. Sube al servidor la entidad especificada (archivo) en la URI. Si la URI se refiere a un recurso ya existente, la entidad adjunta en la petición debe considerarse una modificación de la existente en el servidor. Si por el contrario la URI no apunta a ningún recurso existente, se podrá dar opción a que se cree el recurso. Algunas de las posibles respuestas de éxito por parte del servidor a esta petición son:

- 201 (creado): El recurso se ha creado correctamente.
- 200 (ok): El recurso ha sido modificado correctamente.
- 204 (sin contenido): respuesta sin contenido.

Ejemplo: PUT /path/filename.html HTTP/1.1

Ref: https://tools.ietf.org/html/rfc2616#page-55

DELETE

RFC 2616. Borra el recurso especificado por la URI. Las respuestas exitosas posibles son la siguientes:

- 200 (ok): La respuesta incluye una entidad describiendo el estado.
- 202 (aceptada): Si la acción aún no ha sido llevada a cabo.
- 204 (sin contenido): Si la acción se ha llevado a cabo, pero la respuesta no incluye entidad.

Ref: https://tools.ietf.org/html/rfc2616#page-56

HEAD

RFC 2616. Pide una respuesta idéntica a la que correspondería a una petición GET, pero en la petición no se devuelve el cuerpo. Esto es útil para poder recuperar los metadatos de los encabezados de respuesta, sin tener que transportar todo el contenido.

Ref: https://tools.ietf.org/html/rfc2616#page-54

2.2 HTML, CSS y AngularJS

Estos lenguajes, encargados de mostrar la interface gráfica y darle funcionalidad, van estrechamente relacionados y por ello se describen en el mismo capítulo. Primero se describen de manera muy breve por separado y, a continuación, se ilustra el modo de relacionarse.

HTML

El lenguaje HTML (*HyperText Markup Language* o lenguaje de marcas de hipertexto) es el estándar para la creación de páginas web; la definición y ampliación del cual está a cargo del Consorcio WWW o *World Wide Web Consortium* (W3C).

La gran mayoría de navegadores web han adoptado e interpretan el lenguaje HTML; a medida que van leyéndolo generan los objetos de la interface gráfica, ligados a las etiquetas propias del lenguaje, para presentar información a los usuarios en forma de web.

Al igual que el lenguaje XML, el HTML es una ampliación del SGML y se basa en una estructura jerárquica de etiquetas rodeadas entre corchetes angulares (<, >, /) con atributos y contenido opcionales, de manera que forman elementos HTML; cada elemento, normalmente formado por una etiqueta de inicio y otra de fin, tiene una representación gráfica dentro del navegador web.

Ilustración 2-2

Una página web básica bien definida debe tener la siguiente estructura:

```
<!DOCTYPE html>
<html>
<head>
        <title>Título de página</title>
</head>
<body>
        <h1>Mi primer Encabezado</h1>
        <p>Mi primer párrafo.</p>
</body>
</html>
```

Donde cada etiqueta del documento tiene su propia función, como se describe a continuación:

- DOCTYPE: Define que el documento es de tipo HTML
- <html></html>: Describe un documento HTML
- <head></head>: Proporciona información sobre el documento
- <title></title>: Proporciona un título para el documento
- <body></body>: Describe el contenido visible de la web
- <h1></h1>: Describe un encabezado
- : Describe un párrafo

Todas las etiquetas que se ubican dentro del contenido del elemento <body></body> son opcionales y dependen de la estructura web a crear. Unas de las más relevantes por su gran aplicación, es la siguiente:

- *<a href="http://www.example.com" title="Ejemplo" target="_blank" > Ejemplo<a>*: Describe un hipervínculo o enlace hacia fuero o hacia dentro del sitio web mediante el atributo href. El navegador genera una petición de tipo GET del protocolo HTTP al recurso ubicado en la URI especificada por la variable del atributo href.
- *<form method="POST" action="example.php"></form>*: Describe un formulario dispuesto para ser enviado, mediante un botón de envío – input de tipo *submit*, hacia la URI especificada en el atributo *action* y el método HTTP especificado en el atributo *method*; con la información a enviar presente en el cuerpo del elemento formulario, almacenada en elementos *<input>* de varios tipos.

Hay otras etiquetas opcionales que se ubican dentro del contenido del elemento <head></head> y que aportan información extra sobre varios aspectos se describen a continuación:

- <link>: Crea vínculos hacia hojas de estilo CSS (*Cascade Style Sheet*), descritas en el siguiente apartado, o hacia iconos
- <style></style>: El contenido del cual es de tipo CSS directamente incrustado
- <meta>: Describe metadatos de la web de varios tipos como; autor, título, fecha, palabras clave, descripción, etc.

Finalmente, para ofrecer interacción, con el usuario, a la web y dentro del mismo navegador, existe una etiqueta específica que permite añadirla:

- <script>: Incrusta un script en la web, o llama a uno mediante la inclusión de la URI del recurso en el atributo *src* y la inclusión del tipo mime, por ej. "text/javascript", en el atributo *type* (opcional dependiendo de la versión del protocolo HTTP). Esta etiqueta es la clave para formalizar la relación entre HTML y AngularJS, ya que; al tratarse de código JavaScript, puede ser incrustado en la web y dispuesto para ser utilizado de la manera que se muestra en el apartado: AngularJS.

CSS (Cascade Style Sheets)

Las hojas de estilo en cascada (CSS) describen de qué manera los elementos HTML deben presentarse en el navegador. El consorcio WWW es el encargado de especificar el estándar: se basa en un conjunto de reglas de aplicación a páginas escritas con lenguaje de marcas, para darles formato mediante unas propiedades de color, forma y posición, entre otras. Actualmente, la última especificación es la CSS3 y está dividida por módulos como son "Selectores", "Espacios de nombres", "Color", etc.

En el caso de páginas HTML, el lenguaje CSS se puede utilizar de varias formas, de las cuales las tres más usadas y conocidas son las siguientes:

- Añadiendo propiedades CSS en el atributo *style* de cualquier etiqueta HTML, por ej. *<body style="background-color: green"></body>*
- Añadiendo un elemento *style,* con las etiquetas *<style></style>,* dentro de la página HTML y con código CSS en el contenido del elemento creado.
- Añadiendo un elemento de tipo *link,* dentro del contenido del elemento *head*, y en el atributo *src* se añade la URI del documento de tipo CSS que contiene las reglas y propiedades a aplicar.

El concepto principal que usa el lenguaje CSS a la hora de definir propiedades de un elemento HTML es el concepto de caja (CSS *Box model*): Se basa en considerar cada elemento HTML como envuelto en una caja; con margen, borde, relleno (*padding*) y el contenido. A su vez, los elementos de la caja se pueden subdividir en los cuatro lados para darles formato de manera individual como se ve en la siguiente ilustración.

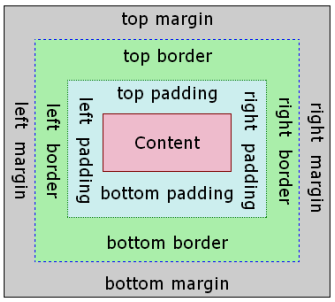


Ilustración 2-3

Dada la gran cantidad de propiedades CSS y la amplia bibliografía que existe sobre ellas, sólo se comenta a modo de ilustración del lenguaje. Existen propiedades que definen un concepto muy específico y otras que permiten agrupar varios conceptos. Dependiendo del resultado buscado, habrá que usar una definición u otra:

- *border: border-width border-style border-color*; → Aplica a los cuatro bordes la anchura, estilo y color especificados de la misma forma.
- *border-color: border-color-top border-color-right border-color-bottom border-color-left;* → Aplica a los cuatros bordes el color indicado en su posición; en sentido de las agujas del reloj y empezando por el de arriba.

En el módulo “Selectores” se especifican las distintas reglas que, como el nombre del módulo indica, seleccionan los distintos elementos HTML para aplicarles las propiedades que haya definidas. Los tres selectores básicos del lenguaje son los siguientes:

- .class: Selecciona todos los elementos que tengan el nombre de la clase en el atributo *class*, por ejemplo: * * se selecciona con el patrón *.resumen*, y a su vez se seleccionan todos los otros elementos HTML que tengan como variable *resumen* en el atributo *class*.
- *#id*: Selecciona el elemento que tiene como atributo id el valor especificado en el selector, por ejemplo: * * se selecciona con el patrón *#resumen* y sólo selecciona este elemento; una página web bien formada no admite dos elemento con atributo *id* idéntico.
- *element*: Selecciona todos los elementos de tipo especificado en el selector, por ejemplo: * * se seleccionan con el selector *p*; además, si existen más elementos de tipo *p*

también serán seleccionados y, por consiguiente, se les aplicarán las propiedades especificadas para el selector.

Con estos tres selectores se puede observar la gran potencia que tiene el lenguaje para seleccionar distintos elementos HTML dentro del documento. Además, cabe mencionar que; estos y otros selectores se pueden combinar para seleccionar de forma más específica los elementos de un mismo documento HTML. Por ejemplo, si se tiene la siguiente estructura:

```
<div id="menu1"><p class="first">Menú 1 principal</p></div>
<div id="menu2"><p class="first">Menú 2 principal</p></div>
```

Los siguientes selectores dan resultado distinto:

- p → selecciona los dos elementos *p*
- p.first → selecciona los dos elementos *p*, dado que los dos tienen como atributo *class* la variable *first.*
- *#menu1 p.first* → selecciona sólo el elemento *p* que tiene como atributo *class* la variable *first* y, además, se encuentra dentro del elemento con *id=”*menu1”.

Con estos lenguajes se pueden crear interfaces muy potentes y profesionales, pero, muchas veces, la gran dificultad para un programador no reside en la complejidad de utilización del lenguaje, sino en la capacidad para crear/diseñar interfaces profesionales a nivel de visualización y combinación de colores. Es por ello que existen herramientas HTML y CSS como es el caso de Bootstrap; que es un *framework* de HTML y CSS que ayuda en la creación y facilita el diseño porque ya viene implementado en los elementos HTML; simplemente, hay que seguir cierta estructura HTML de clases y elementos para conseguir crear buenas interfaces gráficas, gracias al conjunto de propiedades CSS que el entorno Bootstrap proporciona. Se puede ver cómo usarlo y qué características reúne en su página web, http://getbootstrap.com/.

AngularJS

AngularJS o directamente Angular es lenguaje de código abierto y está siendo mantenido por Google. Su utilización principal es dar funcionalidad a aplicaciones web de una sola página o SPA (*Single Page Aplications*), patrón que se comentará en su sección correspondiente del capítulo de diseño de la aplicación. El *framework* AngularJS de JavaScript, actualmente es muy conocido y es ampliamente utilizado para dar funcionalidad a la web desde el mismo navegador; dado que es un código de scripting y que el navegador es uno de sus intérpretes, es un tipo de código que se conoce como “código cliente” para referirse a que el navegador lo ejecuta a medida que lo va leyendo (propiedad de todo código JavaScript).

Otra de las características del *framework* AngularJS, también nativa del lenguaje JavaScript, es que trabaja directamente con objetos. Usa objetos propios y creados, a su vez, con objetos de tipo factoría, como es el caso del objeto global *angular.Module*, que es de tipo interfaz, y es sobre el cual reside y envuelve la aplicación Angular; con los distintos componentes tales como servicios y controladores, entre otros.

Los servicios son un ejemplo de tipo de objetos propios de Angular, que se pueden compartir entre controladores; ya que, como su nombre indica, ofrecen un servicio que puede ser necesario para compartir funcionalidad en las distintas partes de la web. De hecho, entre los expertos de Angular, el uso de los servicios se considera una buena práctica a la hora de compartir, también, variables entre objetos.

Los controladores, también tipo de objetos propios de Angular, se encargan de dar funcionalidad a partes concretas de la aplicación y separar contextos de variables, dentro del documento HTML, mediante otros objetos de AngularJS nombrados "scope", que unen el controlador con la vista.

La instrucción de Angular para generar el objeto principal es la siguiente:

```
var myApp = angular.module('myApp', []);
```

La interfaz *angular.Module* tiene varios métodos para generar más objetos propios del *framework*, a parte de la forma nativa del lenguaje – *new* – de JavaScript; tales como objetos de servicio y objetos de tipo controlador, que se crean con los métodos *angular.Module.service* y *angular.Module.controller* respectivamente. Un ejemplo de creación de estos objetos sería el siguiente:

Siguiendo la estructura de la llamada al método de creación del servicio, service(name, constructor):

```
myApp.service('myService', function() {
    this.myFunc = function (x) {
        return x*x + x + 2;
    }
});
```

Donde se puede observar la creación del servicio con nombre *myService* y el parámetro constructor del mismo; que se trata de la asignación de una función a la propiedad del servicio llamada *myFunc*. Este servicio puede ser utilizado mediante la inyección de dependencia de Angular, como se ilustra a continuación y siguiendo la estructura de la llamada al método de creación del objeto controlador, *controller(name, constructor)*:

```
myApp.controller('myCtrl',['$scope', myService', function($scope,
myService) {
    $scope.myRes = myService.myFunc(24);
}]);
```

Donde se ve la creación del objeto con nombre *myCtrl* y su propiedad *myRes*; la cual, para obtener su valor hace uso del recién creado y, correctamente inyectado, servicio. Además, cabe destacar el uso del objeto *$scope* de Angular, ya que otorga al controlador una propiedad muy útil y, a la vez, trivial de implementar gracias a la inyección del mismo: Cualquier propiedad que se define en el controlador, asociada este objeto, puede ser accesible en cualquier parte de dentro del elemento del documento HTML donde se define el controlador, mediante directivas Angular, ilustradas en el siguiente apartado. Dicho controlador está listo para ser incrustado y utilizado en una página HTML con la finalidad de proveer funcionalidad, como se verá en el siguiente apartado.

Antes de pasar al siguiente apartado, cabe mencionar una propiedad muy importante que deriva de la forma de trabajar de JavaScript; directamente con objetos y sin la definición propia de clase, y es la capacidad de generar objetos dinámicamente y en tiempo de ejecución. En este proyecto, esta propiedad es de gran ayuda para crear objetos complejos desde cadenas de caracteres en formato JSON (*JavaScript Object Notation*) recibidas desde el *back-end* de la aplicación; gracias al uso de inicializadores de objetos de Javascript, donde más adelante se explica en detalle su uso concreto.

Extensión del HTML con Directivas y Expresiones de AngularJS

A fin de ilustrar cómo interactúan estos dos lenguajes de manera muy básica, se debe introducir el concepto de Angular para extender el HTML: Directivas.

Las directivas de Angular son una herramienta que provee funcionalidad mediante la inserción de las mismas dentro del código HTML. Angular da la posibilidad de crear directivas propias; aunque también ofrece sus directivas nativas, en forma de atributos HTML. Normalmente, las directivas nativas de Angular empiezan por el prefijo *ng-* y, para convertirlas en código HTML5 válido, se les añade el prefijo *data-* por delante del anterior. Por ejemplo, la directiva *ngApp* se usaría de la siguiente manera: *data-ng-app*.

Una de las directivas principales es la directiva *ngModel*, cuya función es la de vincular el valor de los elementos HTML, tales como *input, select, textarea*, etc. a una propiedad del objeto *scope* mediante el uso de un controlador.

Así, por ejemplo, teniendo en cuenta la definición del controlador en el apartado anterior – AngularJS; el uso de dicho controlador, su propiedad asociada y la directiva

que se acaba de describir se usarían de la siguiente manera – dentro de un documento HTML:

```
<body data-ng-app=”myApp”>
        <div data-ng-controller=”myCtrl”>
                <input type="text" ng-model="myRes">
        </div>
</body>
```

Donde se puede ver que la propiedad *myRes* del controlador *myCntrl* está asociada al elemento *input* de tipo texto, que se encuentra dentro del ámbito del controlador y, significa que: si la propiedad *myRes* varía su valor, entonces, mediante la intervención del objeto *scope*, el valor del input se modificará en consecuencia, y viceversa.

Se puede entrever el patrón de programación que siguen estos elementos: Los elementos de HTML y CSS, los objetos *scope* y controlador de Angular, una variante del Modelo Vista Controlador – MVC en el cliente, como es MVVM. En el capítulo de MVVM se describirá dicho patrón.

2.3 API RESTful

La Transferencia de Estado Representacional REST (del inglés, *Representational State Transfer*) es un estilo de arquitectura software para sistemas hipermedia distribuidos, como la *Wold Wide Web*. El primero que introdujo el término fue Roy Fielding en su tesis doctoral, en el año 2000 y se basa en las siguientes seis restricciones:

- Interfaz uniforme: La uniformidad de la interfaz entre cliente y servidor se basa en los siguientes cuatro principios:
  - Basada en recursos: Conceptualmente, los recursos y la representación de los mismos que se envía al cliente son cosas distintas. Por ejemplo, el servidor no envía su base de datos al cliente sino una representación JSON, HTML o XML, etc. de ciertos registros.
  - Manipulación de recursos a través de representaciones: Cuando un cliente tiene una representación de un recurso, incluidos sus metadatos, tiene suficiente información para modificar o eliminar el recurso del servidor.
  - Mensajes auto-descriptivos: Cada mensaje incluye suficiente información para su correcto procesado.
  - Híper-medios como motor de estado de la aplicación o HATEOAS (del inglés, *Hypermedia as the Engine of Application State*): Cuando es necesario, el servidor devuelve, en el cuerpo o en la cabecera del mensaje, enlaces con la URI de recursos relacionados con el de la petición.

- Sin estado: Usa el protocolo HTTP para el envío de mensajes, cuya información es suficiente para procesar la petición.
- Cacheable: Las respuestas del servidor deben definir implícitamente o explícitamente si son cacheables o no, para prevenir a los clientes reusar datos inapropiados o viciados a peticiones futuras.
- Cliente-Servidor: La interfaz que separa al cliente del servidor es uniforme. Esta separación implica que los servidores y los clientes se pueden reemplazar y desarrollar independientemente mientras la interfaz se mantenga inalterada.
- Sistema a capas: Un cliente no puede saber si está directamente conectando con un servidor final o con un intermediario del tipo balanceador de carga, puerta de enlace, etc. cuya función puede ser mejorar la escalabilidad del sistema, reforzar las políticas de seguridad, etc.
- Código sobre demanda: Los servidores pueden transferir lógica a los clientes a través del envío de scripts ejecutables en el cliente, como es el caso de JavaScript o Java *applets*. Esta restricción es opcional.

Si un servicio cumple las seis restricciones anteriores, a excepción de la última que es opcional, se considera un servicio RESTful.

El uso que se le da, en el proyecto, es la creación de una API para que las interfaces gráficas de usuarios puedan interactuar con los datos de la base de datos, mediante representaciones de los objetos en formato JSON.

2.4 Programación orientada a objetos con Java

La programación orientada a objetos es un modelo de programación basado en la estructuración del código en conceptos llamados “objetos”; los cuales poseen un conjunto de propiedades o atributos – cuyo valor forma el estado del objeto, y un conjunto de procedimientos o métodos – cuya función suele ser la modificación de su estado.

Clases, Interfaces y herencia

En Java, los objetos son instancias de clases, es decir; las clases definen el comportamiento de los objetos y, al menos un método llamado constructor, para crear/generar/instanciar los objetos a partir de la clase. Internamente, Java trata los objetos como variables complejas, ya que pueden contener varios tipos de datos o incluso métodos, en forma de puntero; apuntan a direcciones de memoria, donde se almacena el contenido de las mismas.

Un ejemplo de clase, que reúne las características para poder instanciar objetos de ella sería la siguiente:

```
public class User {

    public String name;
    public String surname;

    public User (String name, String surname) {
        this.name = name;
        this.surname = surname;
    }

    public void changeName (String newName) {
        this.name = newName;
    }

}
```

Mediante el operador *new* de Java se podrían instanciar objetos de la clase anterior, como se muestra a continuación:

```
User user = new User (“Frodo”, ”Bolsón”);
```

Existen también las interfaces, que son tipos de clases a partir de las cuales no está permitido instanciar objetos. Se utilizan para definir métodos, ya que ninguno está implementado, es decir; son todos abstractos y deben implementarse en sub-

clases para poder instanciar objetos a partir de ellas. Una posible interface sería la siguiente:

```
public interface Door {

    public boolean opened;

    public void toggleDoor ();

}
```

Y una clase que implementase la interface anterior debería contener la palabra clave *implements* y se vería como sigue:

```
public class DoorImpl implemments Door {

    public DoorImpl() {
        this.opened = false;
    }

    public void toggleDoor () {
        opened = !opened;
    }

}
```

La implementación de una interfaz es un tipo de herencia que permite Java, concepto que, a su vez, ayuda a desarrollar la idea de polimorfismo.

Polimorfismo

Una variable de súper-clase puede apuntar a cualquier objeto que herede de ella en tiempo de ejecución; incluso puede apuntar a más de uno, no simultáneamente pero sí secuencialmente, en una misma ejecución. Y, con implementaciones distintas de un método en cada clase; al ejecutar la sentencia de llamada al método, ejecutará las acciones especificadas en las secuencias de código de la implementación a la que apunte.

Tipos de datos

A parte de los conceptos de objetos y clases, Java es un lenguaje fuertemente tipado, y los tipos de datos primitivos son los siguientes:

- *byte*: Variable de tipo entero que ocupa 1 byte en memoria. Su rango de valores va desde el -128 al 127.
- *short*: Variable de tipo entero que ocupa 2 bytes en memoria. Su rango de valores va desde el -32,768 a 32,767

- *int*: Variable de tipo entero que ocupa 4 bytes en memoria. Su rango de valores es desde -2,147,483,648 a 2,147,483,647.
- *long*: Variable de tipo entero que ocupa 8 bytes en memoria. Su rango de valores es desde aprox. -9.2×10^{18} a aprox. 9.2×10^{18}.
- *float*: Variable de tipo decimal que ocupa 4 bytes en memoria. Su rango de valores es desde aprox. -3.4×10^{38} a aprox. 3.4×10^{38}.
- *double*: Variable de tipo decimal que ocupa 8 bytes en memoria. Su rango de valores es desde aprox. -1.79×10^{308} hasta aprox. 1.79×10^{308}.
- *char*: Variable que almacena un carácter simple y ocupa 2 bytes en memoria, ya que internamente trabaja con Unicode.
- *boolean*: Variable de tipo verdadero o falso que ocupa 1 byte en memoria.

Tener conocimiento de los datos primitivos ayuda al programador a la hora de escoger qué tipo usar en cada caso; dependiendo del rango de valores que la variable pueda tomar, sin exceder ni sobreestimar el rango posible de la variable respecto del tipo de datos que se escoja.

2.5 Spring Framework

Estructura básica

Spring es un *framework* de Java que provee de una infraestructura para ayudar a desarrollar aplicaciones a partir de objetos de Java básicos, o POJOs (del inglés, *Plain Old Java Objects*), incluyendo servicios de empresa.

El funcionamiento de Spring se basa en la generación de objetos a los que llaman *beans*, a partir de configuración propia y de objetos de java básicos. Para relacionar los *beans*, entre ellos, Spring ha creado el concepto de inyección de dependencia mediante la inversión de control, internamente con el uso de la Reflexión de Java. Asimismo, con el mismo mecanismo, Spring permite implementar patrones de diseño formalizados que se pueden integrar, de manera sencilla, usando la configuración de *beans*, en la aplicación. Dicha configuración se permite describir en ficheros XML, mediante clases de configuración o con el uso de anotaciones de Java en clases concretas. Éste último tipo de descripción es el escogido en el proyecto por simplicidad y orden, ya que las configuraciones de cada clase se encuentran en la misma y permite al programador presente y/o futuro observar qué rol o patrón implementa una vez procesada por Spring. En la siguiente ilustración se puede observar cómo se comporta Spring para generar los *beans* a partir de configuración y clases estándar.

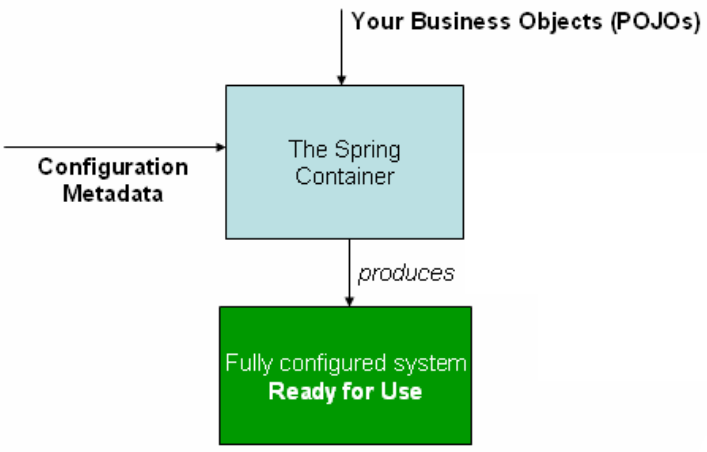


Ilustración 2-4

Actualmente, el *framework* está modulado en unos 20 módulos, compuestos por *JARs* y agrupados por conceptos, que aportan distintos servicios, de manera que sólo incluyas en el *classpath* los que la aplicación requiera. Algunos de los usados para este proyecto son:

- Data *Access/Integration*: Contiene el módulo *spring-jdbc*, que provee una capa de abstracción que elimina la codificación tediosa de JDBC.
- Web: Entre otros, el grupo de web contiene el módulo *spring-webmvc* que ofrece los servicios de Spring para el patrón *model-view-controler* (MVC) y para REST.
- Core: Contiene dos de los módulos fundamentales del *framework*; el *spring-core* y el *spring-beans* que ofrecen las funciones de inversión de control (IoC) y la inyección de dependencia. Además, *BeanFactory* es una implementación bastante sofisticada del patrón factoría; elimina la necesidad de programar el patrón *Singleton* y, permite desacoplar la configuración y la especificación de dependencias de la lógica de programación, mediante el uso de anotaciones y/o ficheros XML de configuración.

Spring *framework* contiene una parte específica para web; Spring Web MVC *framework*; está diseñado alrededor de un *Servlet* central – *DispatcherServlet* – que hace el papel de "*Front Controller*" y envía las peticiones que recibe del navegador hacia unos *handlers* o controladores. A parte, ofrece funcionalidades que facilitan el desarrollo de aplicaciones web.

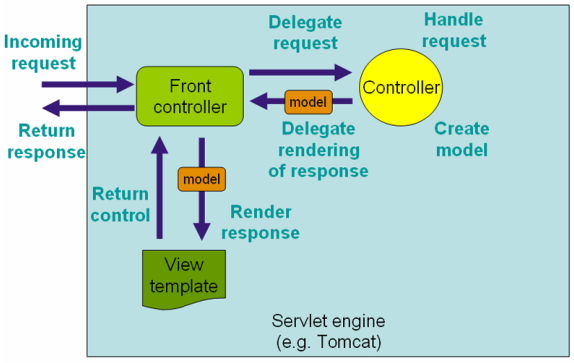


Ilustración 2-5

Los *handlers* por defecto de Spring son las clases de tipo controlador, que se especifican mediante la anotación *@Controller* o, *@RestController* para el caso de controlador de tipo REST. Éstos interpretan los datos de la petición, introducidos por el usuario, para poder ser tratados de la manera que el método, anotado con: *@RequestMapping*, *@RequestParam*, etc., tenga implementado. Un ejemplo de *handler* de tipo controlador REST sería el siguiente:

```
@RestController
@RequestMapping ("/users")
public class UsersRestController {

    @RequestMapping (value="/new", method = RequestMethod.GET)
    public User getNewUser () {
        return new User ();
    }
}
```

Donde se pueden observar los siguientes detalles:

- La clase *UsersRestController* está anotada con *@RestController* y *@RequestMapping("/users")*, donde con la primera queda definida como *handler*, que se encarga de gestionar peticiones, para el *DispatcherController* y con la segunda se le asignan todas las peticiones que su *path* empiece por */users*.
- El método *getNewUser()* está, también, anotado con *@RequestMapping* pero la anotación contiene otro parámetro que refina el tipo de petición a la que está mapeado el método; en este caso la petición HTTP debe ser de tipo GET. Además, el *path* de la petición debe contener la parte indicada en la anotación @RequestMapping general del controlador más lo indicado en el parámetro *value*; siendo el *path* compuesto total "/users/new".
- Finalmente, cabe detallar cómo el método devuelve un objeto serializado de clase *User*: En el primer punto de estas observaciones no se ha detallado que la anotación *@RestController* es la composición de las anotaciones *@Controller* y *@ResponseBody*. Es con la primera con la que se indica que la clase hace la función de *handler* y, con la segunda, se indica que todos los métodos devuelven sus respuestas directamente dentro del cuerpo de la respuesta HTTP. Finalmente, con la ayuda de la librería Jackson, la serialización del objeto se filtra a JSON.

Este tipo de controlador REST es el más usado a la hora de implementar APIs de tipo REST, como la que se ha implementado en este proyecto. Además, con la ayuda de las plantillas de URI se pueden llevar a cabo otro tipo de mapeos y, paso de parámetros, como se puede ver en el siguiente ejemplo, que los ilustra junto a otros conceptos:

```
@RestController
@RequestMapping ("/users")
public class UsersRestController {

    private final UserRepository userRepository;

    @Autowired
    public UserRestController (UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @RequestMapping (value="/{userId}", method = RequestMethod.GET)
    public User getUser (@PathVariable int userId) {
        return this.userRepository.findOne(userId);
    }

    @RequestMapping(value = "/{userId}", method = RequestMethod.PUT)
    public User updateUser (@RequestBody User user,
                            @PathVariable Integer userId) {
        User userU = userRepository.save(user);
        return userU;
    }
}
```

En donde se han usado los elementos, aun sin detallar, siguientes:

- En este ejemplo se amplía el abanico de URIs que la anotación *@RequestMapping* puede mapear; donde, en el caso del ejemplo, se puede observar que en el parámetro *value* contiene una plantilla de URI, en formato *String*, con la variable *userId*. El valor de esta variable se puede extraer y asociar al argumento de tipo entero *userId* del método mediante la anotación *@PathVariable*, aplicada al argumento.
- Cuando el controlador del ejemplo atiende a una petición que cumple el patrón de URI compuesto, por ejemplo: "/users/3", se utiliza el repositorio de usuarios, *userRepository*, para extraer de la base de datos el usuario con *userId* igual a 3.

Esta es la estructura básica de interacción entre el navegador web y una aplicación web implementada con el *framework* de Java Spring, para visualizar y modificar información de base de datos. En el siguiente apartado, se detalla el concepto de repositorio de Spring y cómo se relaciona con la tecnología que usa para extraer los datos de la base de datos; pues, hace la función de DAO (del inglés *Data Access Object*) u objeto de acceso a la base de datos.

Como aspecto a tener en cuenta, se puede observar la de/serialización de los objetos de dominio en las peticiones; con simples anotaciones en los métodos del controlador, como por ej. *@RequestBody,* asociadas a los argumentos de objetos del

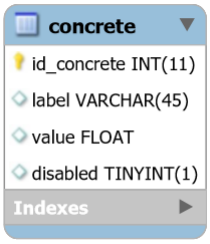
dominio y con la ayuda de la librería *Jackson* para su correcta traducción a/de JSON y XML.

Repositorios de Spring con JPA (Java Persistence Api)

Como se ha visto en el último ejemplo del apartado anterior, los repositorios se usan de intermediarios entre los controladores y la base de datos.

Se basan en la utilización del mapeo de objeto-relacional u ORM (del inglés, *Object Relational mapping*) para generar sentencias SQL que hagan las funciones básicas de CLMB o CRUD (del inglés, *Create, Read, Update and Delete*). El mapeo de objeto-relacional es un modelo de programación para transformar los objetos, en un lenguaje de programación orientado a objetos, en tablas de la base de datos. Dichos objetos son los que se califican con el nombre objetos de dominio.

Un ejemplo básico de mapeo de objeto-relacional para una tabla de base de datos sería el siguiente.




```
@Entity
@Table (name = "concrete", catalog = "structures", schema = "")
public class Concrete implements Serializable {

        @Id
        @GeneratedValue
        @Column (name = "id_concrete")
        private Integer idConcrete;

        @Size (max = 45)
        @Column (name = "label")
        private String label;

        @Column (name = "value")
        private Float value;

        @Column (name = "disabled")
        private Boolean disabled;

        public Concrete () {}
        /* código no relevante omitido */
}
```

Donde se puede ver que el objeto-relacional es una clase normal de Java a la que se le añaden distintas anotaciones con el fin de asociar la clase y sus atributos con el nombre de la tabla y sus campos, respectivamente. Esta manera de relacionar la clase con la tabla es muy poco intrusiva, ya que no hay que modificar el código de la clase para declarar la asociación. Esto es gracias al uso de la persistencia de Java o JPA (del inglés, *Java Persistence Api*), que nos aporta dichas anotaciones con ese fin:

- @*Entity*: Declara que la clase es una entidad de JPA.
- @*Table*: Indica la base de datos y tabla a la cual se asocia la entidad.
- @*Column*: Informa de la columna a la cual va asociado el atributo.
- @*Id*: Indica que el atributo hace la función de clave en la tabla.
- @*GeneratedValue*: Informa que el atributo es auto-incrementado automáticamente.

Una vez definida la relación de los objetos de la base de datos de la aplicación, mediante el uso de las anotaciones de Java con JPA, se puede crear una capa más de abstracción con el uso de repositorios. Dichos repositorios son interfaces planas de Java o POJOs que extienden la interface *Repository* o cualquiera de las subinterfaces, como se puede observar en el siguiente ejemplo, en el que se extiende de la interfaz *JpaRepository*:

```
public interface ConcretesRepository extends
JpaRepository<Concrete,Integer> {

        Optional<Concrete> findByIdConcrete (Integer concreteID);

}
```

La interfaz *JpaRepository* contiene los métodos básicos necesarios para realizar las operaciones de creación, lectura, modificación y borrado de los datos de una tabla general. Entonces, es Spring el encargado de crear los objetos a partir de esta interface y mediante el uso de JPA para crear sentencias SQL compatibles, en el caso de este proyecto, con MySQL, que es el motor de SQL utilizado.

Actualmente, con los repositorios se puede trabajar con sentencias SQL declaradas o, directamente, con métodos que describen mediante palabras clave lo que luego Spring traducirá como sentencias SQL, del motor de base de datos especificado en el fichero de configuración. Por ejemplo, usando MySQL como motor de base de datos, y el método con nombre *findByIdConcrete(Integer idConcrete)*, se traduce a la sentencia SQL: *SELECT * FROM concrete WHERE idConcrete = concreteID;*.

A parte de los aspectos básicos de Spring Framework, hay módulos que proveen configuraciones preestablecidas, patrones y herramientas automáticas de manejo de dependencias que creo conveniente comentar brevemente en los siguientes 3 apartados.

Spring Boot

Es un módulo de Spring que provee la configuración básica para desarrollar y ejecutar una aplicación de empresa con Java, basándose en convenciones de configuración: conceptos de seguridad, contenedor de *Servlets* embebido, métricas de aplicación, configuración externa, etc.

Zuul (Gateway de Servicios)

El *gateway* de servicios proporciona el punto único de acceso y simplifica posibilidades como enrutamiento, mantenimiento de versiones, seguridad, etc. para todo el sistema.

Dado que es el único punto de acceso al sistema, permite la realización de procesos comunes para todas las peticiones y respuestas. Además, al proveer enrutamiento, permite implementar el patrón de micro-servicios para el escalado de la aplicación; realizando una separación de la aplicación en varias micro-aplicaciones, o servicios, que proporcionan funcionalidades comunes. En este proyecto, como se verá en el capítulo que describe el sistema, se ha separado en un micro-servicio todo lo que tiene que ver con la API de estructuras, y otros tres con las UIs del cliente para cada tipo de usuario.

Redis

Es un motor de base de datos en memoria, donde los valores se almacenan en formato diccionario (clave/valor) y se usa, en este proyecto, como un servicio externo a Spring para compartir datos entre aplicaciones o micro-servicios.

Plugin de Maven

El equipo de Spring recomienda el uso de alguna de las herramientas de manejo de dependencias automáticas para la aplicación Java, como son; Maven, Gradle o Ivy. En este proyecto se ha usado Maven por alusiones, ya que el equipo de Spring casi siempre lo usa en sus ejemplos y tutoriales.

Maven basa su configuración en un archivo XML llamado POM (del inglés, *Project Object Model*), el cual permite heredar configuración de otros archivos POM definidos como padres. En este proyecto se ha usado el archivo padre *spring-boot-starter-parent* para heredar todas las dependencias de *Spring Boot* y, de esta manera, simplificar mucho la gestión de dependencias básicas.

Dentro del archivo pom.xml, aparte de incluir los espacios de nombres necesarios para usar las etiquetas oportunas de Maven en formato xml, está la información de todas las dependencias o librerías de la aplicación, contenidas dentro del elemento XML *dependencies*. En el siguiente ejemplo se muestra cómo se incluiría la

dependencia con el módulo de Spring *spring-boot-starter-web*, para que, a la hora de compilar y ejecutar la aplicación, Maven la incluyese en el *classpath*:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

2.6 SQL, MySQL y MariaDB

SQL

SQL (del inglés, *Structured Query Language*) es un lenguaje declarativo (especifica qué hacer, pero no cómo ni en qué orden) de acceso a bases de datos relacionales, estandarizado en 1986 por el ANSI (del inglés, *American National Standards Institute*) y adoptado por la ISO (del inglés, *International Organization for Standardization*) al año siguiente.

Se basa en una fuerte base teórica y está orientado al manejo de conjuntos de registros, permitiendo una alta productividad en codificación y la orientación a objetos. Tiene las características siguientes:

- Lenguaje de definición de datos (LDD): El LDD de SQL se encarga de la modificación de la estructura de la base de datos. Hay cuatro operaciones básicas: CREATE, ALTER, DROP y TRUNCATE.
- Lenguaje interactivo de manipulación de datos (LMD): El LMD de SQL incluye lenguajes de consultas basado tanto en álgebra relacional como en cálculo relacional de tuplas. Ejemplos de sentencias son: SELECT, INSERT, UPDATE y DELETE.
- Integridad: El LDD de SQL incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos.
- Definición de vistas: El LDD incluye comandos para definir las vistas.
- Control de transacciones: SQL tiene comandos para especificar el inicio y el final de cada transacción.
- SQL incorporado y dinámico: Se pueden incorporar instrucciones SQL en lenguajes de programación como JAVA, C, PHP, Cobol, etc.
- Autorización: El LDD incluye comandos para especificar los permisos de acceso a las relaciones y vistas.

MySQL y MariaDB

MySQL es un sistema de gestión de base de datos relacional, desarrollado bajo licencia dual GPL/Licencia comercial por *Oracle Corporation* y está considerada como la base de datos open *source* más popular del mundo.

A diferencia de la mayoría del software con Licencia pública general de GNU, donde el desarrollo del mismo se lleva a cabo por la comunidad pública, los derechos de MySQL están en poder de la empresa privada que lo desarrolla. Es por ello que, en 2009, nace un proyecto paralelo denominado MariaDB con licencia GPL y derivado de MySQL (tiene una alta compatibilidad), donde algunos de los desarrolladores eran originales de MySQL como uno de los fundadores de MySQL Michael (Monty) Widenius.

2.7 MATLAB y MCR

MATLAB (abreviatura de *MATrix LABoratory*) es una herramienta de software matemático propietaria que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje propio de programación; el cual es un lenguaje de cálculo técnico interpretado que permite operaciones de vectores y matrices, funciones, cálculo lambda y programación orientada a objetos.

Dado que MATLAB es un software propietario, el desarrollador está sujeto al entorno de programación o IDE. Aun así, existen dos módulos que permiten la ejecución de scripts sin la necesidad del IDE y como aplicaciones autónomas de libre distribución: el MATLAB *Compiler* y el MCR (del inglés, MATLAB *Component Runtime*).

Mediante el MATLAB Compiler se puede crear una aplicación autónoma ejecutable desde línea de comandos, empaquetándola y encriptándola para la protección de la propiedad intelectual de la misma. Ésta podrá ser ejecutada en el PC que se encuentre si éste tiene instalado el MATLAB, o bien, tiene instalada la versión correspondiente de MCR respecto al compilador usado para crearla.

3. Requerimientos

En este apartado se especifican los requerimientos de la aplicación, los cuales están distribuidos en perfiles de usuario.

3.1 Autenticación

En la página de inicio, cualquier usuario registrado puede autenticarse mediante usuario y contraseña. En un futuro, debe tener la opción de autenticarse mediante un servidor de identidades como, por ejemplo: Google, Facebook; que usan el protocolo más seguro actualmente para este servicio; OAUTH2.

Autenticación con Usuario y Contraseña

El nombre de usuario y la contraseña se escogen en el registro. Pero la contraseña es lo único modificable, una vez escogidos.

Registrarse

Cualquier usuario de internet puede solicitar registrarse. El registro será válido sólo si el usuario con perfil administrador acepta el registro solicitado. Además, el administrador podrá registrar usuarios sin tener que volver a validarlos.

3.2 Opciones de Usuario Básico

Todo usuario capacitado para entrar en la aplicación es tratado como un usuario estándar a no ser que tenga asociado algún otro perfil de usuario de tipo administrador y/o súper administrador.

Calcular Estructura

Este cálculo es la única opción que llama a la aplicación hecha en Matlab, de cálculo de estructuras de sección rectangular. Se detallan todas las especificaciones de la parte principal de la aplicación web, a continuación.

Variables de entrada

Todas las variables de entrada, que se encuentran en la interface gráfica, se refieren a las características físicas y de composición que debe tener la siguiente estructura rectangular (Ilustración 3-1). En todas las variables, la parte decimal se separa de la parte entera mediante la inserción de un punto.

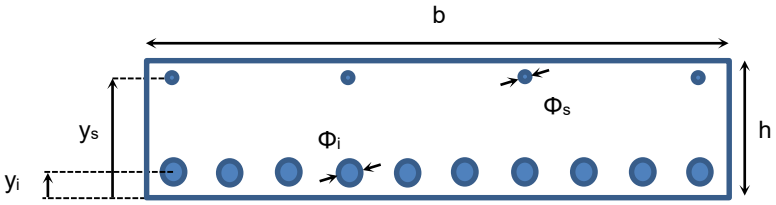


Ilustración 3-1

Ancho de la estructura (b) (*)

Variable numérica **b** $\in \mathbb{R}^+$ → **b**$\in$ (0, ∞). Representa el ancho, en metros, de la estructura rectangular. Es un campo obligatorio.

Canto de la estructura (h) (*)

Variable numérica **h** $\in \mathbb{R}^+$ → **h**$\in$ (0, ∞). Representa la altura, en metros, de la estructura rectangular. Es un campo obligatorio.

Hormigón (f_{ck}) (*)

Variable numérica con valores pre-definidos seleccionables mediante desplegable. El rango de valores, por defecto, será [20, 25, 30, 35, 40, 45, 50, 55, 60, 70, 80, 90, 100], pero el súper usuario puede añadir valores para adaptarlos a las nuevas normativas, en caso de que sea necesario. Representa las distintas resistencias tipificadas en la normativa. Es un campo obligatorio.

Acero (f_{yk}) (*)

Variable numérica con valores pre-definidos seleccionables mediante desplegable. El rango de valores, por defecto, será [B-400, B-500], pero el súper usuario puede añadir valores para adaptarlos a las nuevas normativas, en caso de que sea necesario. Internamente los valores serán [400, 500], respectivamente. Representa las distintas calidades del acero. Es un campo obligatorio.

Fibras (boolean)

Variable 1 Fibras ($f_{R,1}$)

Variable numérica $\mathbf{f_{R,1}} \in \mathbb{R}^+ \rightarrow \mathbf{f_{R,1}} \in (0, \infty)$, $\mathbf{f_{R,1}}$ se aproxima (con un decimal) al múltiplo de 0.5 inmediatamente inferior, siempre que la variable fibras sea true. Si la variable fibras es false entonces $f_{R,1} = 0$.

Variable 4 Fibras ($f_{R,4}$)

Variable numérica $\mathbf{f_{R,4}} \in \mathbb{R}^+ \rightarrow \mathbf{f_{R,4}} \in (0, \infty)$, $\mathbf{f_{R,4}}$ se aproxima (con un decimal) al múltiplo de 0.5 inmediatamente inferior, siempre que la variable fibras sea true. Si la variable fibras es false entonces $f_{R,4} = 0$.

Esbeltez (λ)

Variable numérica $\boldsymbol{\lambda} \in [1,100]$, siempre que la variable fibras sea true. Si la variable fibras es false, $\boldsymbol{\lambda} = 0$.

Armado capa inferior (boolean)

Diámetro barras capa inferior (ϕ_i)

Variable numérica con valores pre-definidos seleccionables mediante desplegable. El rango de valores, por defecto, es: [8, 10, 12, 14, 16, 20, 25, 32], pero el súper usuario puede añadir valores para adaptarlos a las nuevas normativas, en caso de que sea necesario. Representa el valor del diámetro (en mm) de las barras de la capa inferior, que está tipificado en la normativa. Es un campo obligatorio si la variable de armado inferior está activada. El valor por defecto debe ser de 10mm.

Cota del centro de gravedad de la capa inferior (y_i)

Variable numérica $\mathbf{y_i} \in [\ 0 - \mathbf{h}\ ]$. Representa la cota del centro de gravedad (en metros) de la capa inferior de las barras de acero.

Número de barras en la capa inferior (n_i)

Variable numérica entera positiva, $\mathbf{n_i} \in \mathbb{N}^+$. Representa el número de barras de la capa inferior.

Armado capa superior (boolean)

Diámetro barras capa superior (ϕ_s)

Variable numérica con valores pre-definidos seleccionables mediante desplegable. El rango de valores, por defecto, es: [8, 10, 12, 14, 16, 20, 25, 32], pero el súper usuario puede añadir valores para adaptarlos a las nuevas normativas, en caso de que sea necesario. Representa el valor del diámetro (en mm) de las barras de la capa superior, que está tipificado en la normativa. Es un campo obligatorio si la variable de armado superior está activada, y tiene valor 10mm por defecto.

Cota del centro de gravedad de la capa superior (y_s)

Variable numérica $\mathbf{y_s} \in [\ 0 - \mathbf{h}\ ]$. Representa la cota del centro de gravedad (en metros) de la capa superior de las barras de acero.

Número de barras en la capa superior (n_s)

Variable numérica entera positiva, $\mathbf{n_s} \in N^+$. Representa el número de barras de la capa superior.

A parte de las restricciones específicas derivadas de la definición de cada variable, deben cumplirse las siguientes; tanto para el número de barras inferior y superior,

$$\mathbf{n} \leq \frac{\boldsymbol{b} + \boldsymbol{\phi}}{2\,\boldsymbol{\phi}}$$

$$\mathbf{n_s} \neq \mathbf{n_i}$$

como para los centros de gravedad:

$$\mathbf{y_s} > \mathbf{y_i}$$

Resultados de la aplicación de Matlab (presentados sobre web)

Los resultados que genera la aplicación de Matlab, y que luego se representan en formato web, vienen definidos en este sub-apartado, tal como sigue:

Apartado de materiales

Los materiales que componen la sección son el hormigón (en masa o con fibras) y el refuerzo con barras de acero.

El **comportamiento del hormigón** se simula con una función no lineal cuando se trata de tensiones de compresión (-3.5‰ < $\boldsymbol{\varepsilon_c}$ ≤ 0.0), ver ilustración 1.1b. Las tensiones de tracción (0.0 < $\boldsymbol{\varepsilon_c}$ ≤ $\boldsymbol{\varepsilon_3}$), en el caso de un hormigón reforzado con fibras se simulan con un diagrama tri-lineal. Para ello, se emplea el diagrama propuesto por el Model Code 2010 (MC-2010). Los parámetros involucrados son:

f_{cd}

f_{cd} : $\alpha_{cc} \cdot f_{ck} / \gamma_c$; Es el valor de diseño de la resistencia a compresión del hormigón; siendo **f_{ck}** el valor característico de la resistencia a compresión; **γ_c** el coeficiente parcial de seguridad del hormigón y **α_{cc}** el coeficiente de cansancio del hormigón (0.85-1.00).

ε_{co}

Es la deformación del hormigón para la cual se alcanza **f_{cd}**. Se trata de un parámetro que el modelo estima internamente con unas ecuaciones sugeridas en el MC-2010.

σ_u

Es la deformación última cuando se alcanza la deformación de rotura del hormigón sometido a compresiones (**ε_{cu}** = 3.5‰).

σ_i y ε_i

Son los valores de las tensiones y deformaciones, respectivamente, que definen el diagrama trilineal del hormigón reforzado con fibras sometido a tracción.

E_c

Es el módulo de deformación del hormigón.

$f_{R,i}$

Son las resistencias residuales a flexotracción del hormigón reforzado con fibras

Asimismo, el **comportamiento del acero** para las armaduras se simula con un diagrama constitutivo bilineal (-3.5‰ < $\boldsymbol{\varepsilon_s}$ ≤ 10.0‰), ver Ilustración 1.1c. Las tensiones de tracción (0.0 < $\boldsymbol{\varepsilon_c}$ ≤ $\boldsymbol{\varepsilon_3}$), en el caso de un hormigón reforzado con fibras se simulan con un diagrama tri-lineal. Los parámetros involucrados son:

f_{sy}

Límite elástico del acero que depende del tipo de acero.

ε_{sy}

Es la deformación del acero para cuando se alcanza $\mathbf{f_{sy}}$. $\boldsymbol{\varepsilon_{sy}} = \mathbf{f_{sy}}/\mathbf{E_s}$.

E_s

Es el módulo de Young del acero.

Apartado de ELU Flexión

El modelo, utilizando subrutinas internas programadas, obtiene el diagrama de interacción Axil (**N**) – Momento (**M**), así como el diagrama momento (**M**) – curvatura (**χ**). Ambos diagramas son herramientas indispensables para el diseño de estructuras.

Apartado de ELS Fisuración

En este apartado se representa el diagrama momento (**M**) – ancho de fisura (**w**) de la sección rectangular desde valores de **M** = 0 hasta alcanzar la rotura de la sección ($\mathbf{M_u}$).

Opciones de la aplicación Web

Enviar resultados por e-mail

Opcionalmente o, en un futuro, todos los resultados deben poder ser enviados por e-mail, una vez finalizados.

Opcional: Formato de resultados en PDF

El formato de visualización de resultados, por defecto, es la interface gráfica web y, además, formato PDF, descargable.

Opcional: Enviar acuso de recibo de resultados al e-mail

Todos los resultados pueden tener la opción de enviar un e-mail cuando los resultados hayan sido generados.

Histórico de cálculos

Cada usuario puede ver todos sus cálculos, así como los resultados generados.

Cargar cálculos

Además, cada usuario puede cargar cualquier cálculo hecho; para modificar las variables de entrada y volverlo a generar con los nuevos parámetros.

Recalcular estructura

Se permite volver a calcular la estructura con los parámetros de entrada opcionalmente modificados.

Perfil del Usuario

Cada usuario puede ver todos los campos de su perfil. Los campos y su tipo son los siguientes:

Nombre (*)

Variable de tipo texto y 45 bytes de longitud. Indica el nombre del usuario. Necesario para referirnos a el usuario por su nombre (por e-mail, etc.). Es un campo obligatorio.

Apellidos (*)

Variable de tipo texto y 45 bytes de longitud. Indica los apellidos del usuario. Es un campo obligatorio.

Usuario (*)

Variable de tipo texto y 45 bytes de longitud. Indica el nombre de usuario, necesario para entrar a la aplicación. Es un campo obligatorio.

Contraseña (*)

Variable de tipo contraseña y 45 bytes de longitud. Indica la contraseña escogida por el usuario. Campo necesario para entrar en la aplicación. Es un campo obligatorio.

DNI (*)

Variable de tipo texto y 45 bytes de longitud. Indica el número y letra del Documento Nacional de Identidad del Usuario. Es un campo obligatorio.

E-mail (*)

Variable de tipo texto y 45 bytes de longitud. Indica el e-mail completo del usuario. Es un campo obligatorio.

Fotografía

Variable de tipo texto y 300 bytes de longitud. Únicamente, especifica el nombre de la imagen, ya que todas las fotografías del usuario estarán en el mismo directorio.

Género (Sexo)

Variable de tipo texto y 10 bytes de longitud. Especifica si el usuario es un hombre o una mujer.

Configuración de Opciones

Hay una serie de opciones asignadas a cada usuario y, son las siguientes:

Idioma de la aplicación

Cada usuario puede escoger el idioma de entre una lista de idiomas.

E-mail de confirmación de resultados

El usuario puede decidir si recibir o no confirmación cada vez que tenga los resultados del cálculo preparados en la aplicación.

3.3 Opciones de usuario Administrador

Este perfil de usuario está destinado, básicamente a la gestión de usuarios de la aplicación.

Listado de usuarios

Dado que cualquier usuario de internet se puede solicitar el registro desde la página inicial, el administrador tiene la opción de aceptar/denegar dicho registro del usuario. Esta funcionalidad sirve para que no se registren usuarios indeseados/desconocidos.

El administrador puede cargar la información de cualquier usuario para modificar sus datos (incluso inhabilitarles la entrada).

Registrar Usuarios

El administrador tiene la capacidad de registrar usuarios sin tener que validarlos; dado que él mismo es el que se encarga de la validación de los registros.

3.4 Opciones de Súper Usuario o Súper Administrador

Este perfil de usuario está destinado a gestionar las variables seleccionables tanto a nivel de estructura como a nivel de usuario. Se describen a continuación:

Configuración de parámetros del cálculo de estructuras

En este apartado se especifican todos los campos editables correspondientes a las variables de entrada para el cálculo de estructuras.

Editar el campo de hormigón

Se permite al súper usuario añadir y modificar los valores de las distintas resistencias tipificadas en la normativa.

Editar el campo del acero

Se permite al súper usuario añadir y modificar los valores de las distintas calidades del acero, con el fin de adaptarse a la normativa.

Editar diámetros de las barras de acero

Se permite al súper usuario modificar los valores de los distintos diámetros de las barras de acero de las capas inferior y superior, correspondientes al armado.

Configurar parámetros del usuario

Hay distintos parámetros que dan funcionalidad a la web y se gestionan desde este perfil, son los siguientes:

Editar idiomas de usuario

Se permite al súper usuario añadir idiomas posibles que posteriormente se usarán para ofrecer servicios y mensajes personalizados al usuario.

Editar tipos de documentos

Se dará la posibilidad de editar los tipos de documento que un usuario puede introducir, y será validado con el patrón que se especifique en su casilla.

4. Casos de uso

En este apartado se especifican, en detalle y a alto nivel, las distintas actividades que se deberán realizar para llevar a cabo los principales procesos requeridos del sistema a implementar, es decir los casos de uso.

4.1 Solicitando registro

Actor: Usuario de Internet

1. El usuario de internet ingresa en la página de registro.	
2. El usuario de internet cumplimenta correctamente el formulario de registro.	2.1. Si el usuario de internet se deja algún campo obligatorio o no cumple los requisitos de algún campo, se le informa.
3. El usuario de internet envía el formulario de registro.	3.1. Si hay algún campo incorrecto, se deniega el envío y se informa del campo erróneo.
4. El sistema informa al usuario de internet de que próximamente recibirá la resolución de su solicitud.	

4.2 Validando registro

Actor: Administrador

1. El administrador recibe la solicitud de registro del usuario de internet.	
2. El administrador comprueba que la solicitud es correcta.	
3. El administrador acepta la solicitud de registro.	3.1. Si la solicitud no es correcta, la deniega.
4. El sistema informa al usuario de internet que la solicitud ha sido aceptada.	4.1. Si la solicitud no es correcta, el sistema informa al usuario de ello. Se disculpa y le sugiere un e-mail de contacto.

4.3 Registrando usuario

Actor: Administrador

1. El administrador entra en la página de registro de usuarios.	
2. El administrador rellena el formulario de registro de usuario, correctamente.	2.1. Si hay algún error en algún campo del formulario, se indica mediante el uso de AJAX.
3. El administrador envía el formulario.	3.1. Si hay algún campo incorrecto, el sistema deniega el envío del formulario e informa del/los campo/s incorrectos. Se vuelve al paso 2.
4. El sistema informa al administrador que ha insertado el usuario correctamente.	
5. El sistema envía el usuario y la contraseña adjudicados al e-mail correspondiente al usuario registrado.	

4.4 Modificando perfil de usuario

Actor: Usuario básico

1. El usuario básico entra en su perfil de usuario. (Obteniendo Usuario)	
2. El usuario básico modifica algún campo de su perfil.	2.1. Si hay algún error en los campos modificados del formulario, se indica mediante el uso de AJAX.
3. El usuario básico envía el formulario de modificación del perfil de usuario.	3.1. Si hay algún campo incorrecto, el sistema deniega el envío del formulario e informa del/los campo/s incorrecto/s. Se vuelve al paso 2.
4. El sistema informa que ha modificado el perfil de usuario, correctamente.	

4.5 Obteniendo usuario

Actor: Usuario básico

1. El usuario básico entra en la página del perfil de usuario.	
2. El sistema muestra el perfil de usuario.	

4.6 Modificando cualquier perfil de usuario

Actor: Administrador

1. El administrador usa el caso de uso: Obteniendo Listado de Usuarios.	
2. El administrador escoge un usuario y entra en su perfil.	
3. El administrador modifica algún campo de perfil del usuario escogido.	3.1. Si hay algún error en algún campo del formulario, se indica mediante el uso de AJAX.
4. El administrador envía el formulario de modificación del perfil de usuario.	4.1. Si hay algún campo incorrecto, el sistema deniega el envío del formulario e informa del/los campo/s incorrecto/s. Se vuelve al paso 3.
5. El sistema informa al administrador que ha modificado el perfil del usuario, correctamente.	
6. El sistema informa de la modificación del perfil al usuario correspondiente.	

4.7 Obteniendo listado de Usuarios

Actor: Administrador

1. El administrador entra en la página del listado de usuarios.	
2. El sistema muestra el listado de usuarios.	

4.8 Entrando al sistema

Actor: Usuarios de la aplicación

1. El actor entra en la página inicial.	
2. El actor rellena el formulario de entrada a la aplicación.	
3. El actor envía el formulario rellenado.	3.1. Si hay algún campo incorrecto, p.ej. contraseña vacía, el sistema no deja enviar el formulario e informa de ello.
4. El sistema muestra la página de bienvenida.	4.1. Si la dupla usuario-contraseña no es correcta, el sistema no re-direcciona a ninguna página e informa de la incorrección. Se vuelve al paso 2.

4.9 Obteniendo Cálculo de Estructura

Actor: Usuario básico

1. El usuario básico usa el caso de uso: Obteniendo histórico de cálculos.	
2. El usuario básico presiona sobre el cálculo deseado.	
3. El sistema muestra el cálculo escogido.	

4.10 Calculando estructura

Actor: Usuario básico

1. El usuario básico usa el caso de uso: Obteniendo histórico de cálculos.	
2. El usuario básico presiona el botón de "Nueva Estructura".	
3. El usuario básico rellena los campos correspondientes a las variables de entrada para el cálculo de la estructura.	3.1. Si hay algún error en algún campo del formulario, se indica mediante el uso de AJAX.
4. El usuario básico envía el formulario del cálculo de estructura.	4.1. Si hay algún campo incorrecto, el sistema deniega el envío del formulario e informa del/los campo/s incorrecto/s. Se vuelve al paso 2.
5. El sistema informa al usuario básico que la petición ha sido puesta en la cola de peticiones y enviará un e-mail cuando se haya procesado correctamente.	
6. El sistema procesa el cálculo con Matlab.	
7. El sistema envía el e-mail de confirmación de cálculo procesado.	7.1. Si hay error en Matlab, el sistema debe informar del error.
8. El usuario usa el caso de uso: Obteniendo cálculo de estructura.	

4.11 Obteniendo histórico de cálculos

Actor: Usuario básico

1. El usuario básico entra en la página de histórico de cálculos.	
2. El sistema muestra la lista de cálculos.	

4.12 Recalculando Estructura

Actor: Usuario básico

1. **El usuario básico usa el caso de uso: Obteniendo cálculo de estructura.**
2. **El usuario usa el caso de uso: Calculando estructura, desde el paso 3.**

4.13 Generando Resultados PDF

Actor: Usuario básico

1. **El usuario básico usa el caso de uso: Obteniendo cálculo de estructura.**
2. **El usuario básico presiona sobre el botón de generar PDF de la estructura.**
3. **El sistema genera un archivo PDF descargable del resumen de la estructura calculada.**

4.14 Recibiendo Resultados PDF al e-mail

Actor: Usuario básico

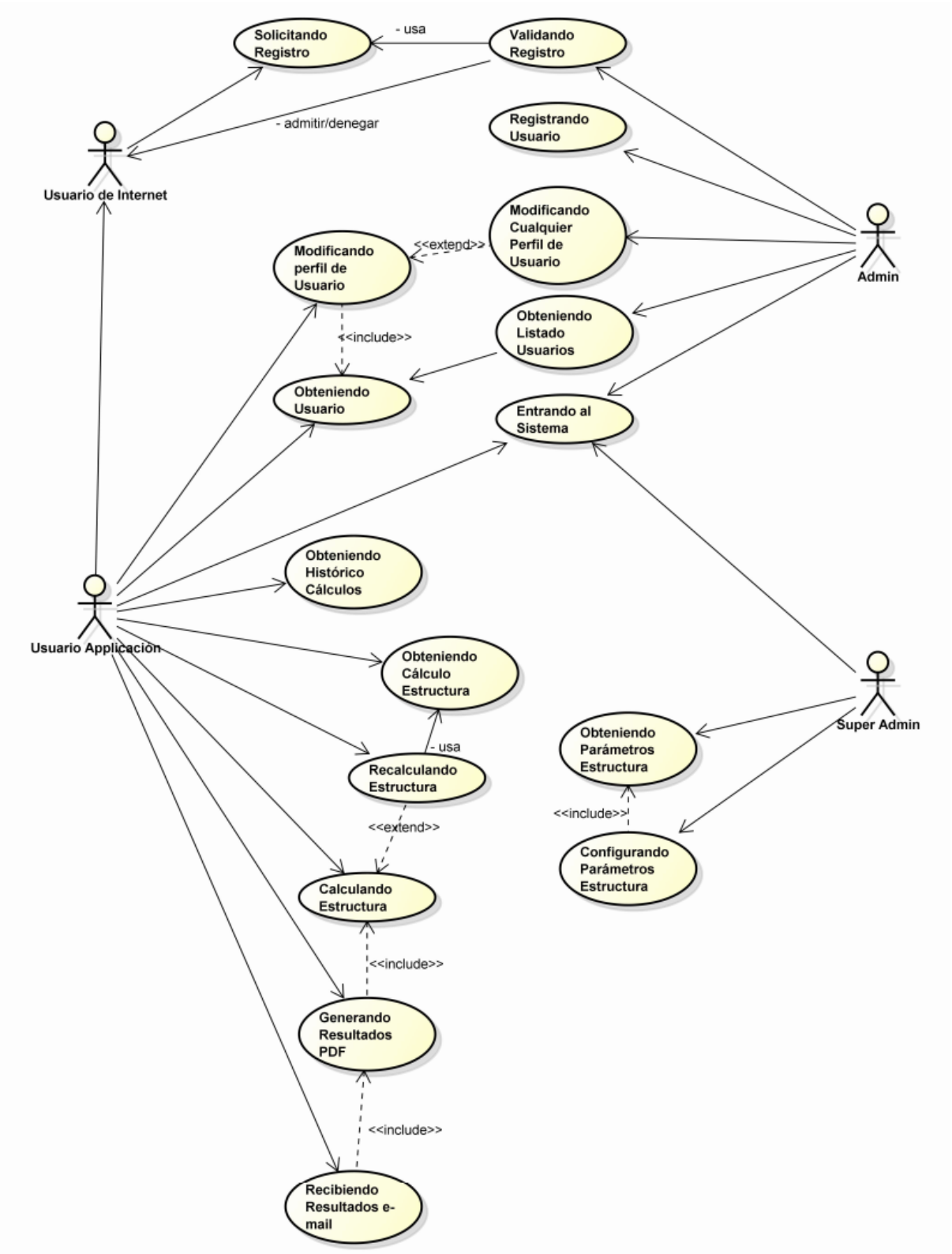
1. **El usuario básico usa el caso de uso: Obteniendo cálculo de estructura.**
2. **El usuario básico presiona sobre el botón de enviar PDF de la estructura por e-mail.**
3. **El sistema genera un archivo PDF del resumen de la estructura calculada y lo envía al e-mail del usuario básico actual.**

4.15 Configurando Parámetros

Actor: Súper Usuario

1. El súper usuario abre la página de configuración de parámetros.	
2. El sistema muestra los parámetros configurados.	
3. El súper usuario añade y/o modifica el/los parámetro/s configurado/s.	
4. El súper usuario guarda los cambios hechos, enviando el formulario de parámetros configurables.	4.1. Si hay algún campo configurado incorrecto el sistema lo indica para que sea corregido.
5. El sistema informa de la correcta modificación de los parámetros de configuración.	

4.16 Diagrama de casos de uso:



5. Diseño de la Aplicación

En este capítulo de diseño se ilustran los distintos aspectos que se han tenido en cuenta a la hora de diseñar el sistema; empezando por los patrones de diseño de software utilizados para solucionar los problemas básicos de estructura, tanto del sistema global como de las aplicaciones concretas que forman el sistema. Les siguen el diagrama y explicación de la estructura completa del sistema utilizada, el modelo de datos implementado para dar soporte a las aplicaciones del sistema y, finalmente, los diagramas de secuencia de los procesos más relevantes.

5.1 Patrones de diseño

MVVM y SPA (Single Page Application)

Estos dos conceptos se comentan en el mismo apartado dado que van íntimamente relacionados y se refieren a la estructura de cada aplicación que forma el sistema.

El patrón de diseño software MVVM (del inglés, *Model View View-Model*) deriva del conocido patrón MVC (del inglés, *Model View Controller*) y, como todo patrón de diseño de software describe una posible solución aplicable a un problema ya conocido y solucionado anteriormente. El patrón MVVM facilita la separación de desarrollos entre la interfaz gráfica o *front-end* y la lógica de negocio o *back-end*. Básicamente, separa la aplicación en tres partes: la vista (*view*), el modelo o modelo de datos (*model*) y el modelo de la vista (*view model*).

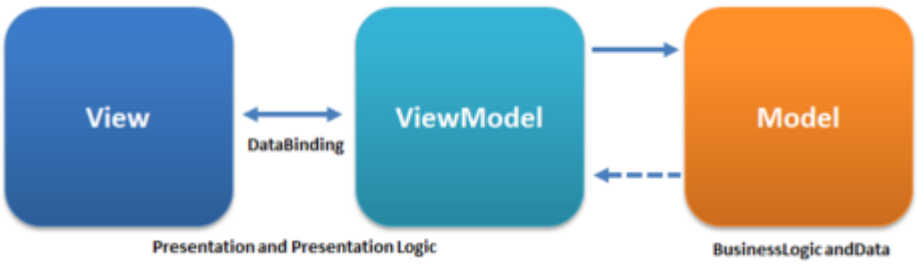


Ilustración 5-1

Donde la vista no varía respecto al patrón MVC en el que la interfaz de usuario, formada por códigos HTML5 y CSS3, corresponde a la parte de la vista en el patrón MVVM. La parte del modelo de la vista (*ViewModel*), dentro del patrón MVVM, se ubica en el código AngularJS y se encarga de: interaccionar con la vista exponiendo métodos y propiedades que forman su propio estado a través de los controladores de Angular y la ayuda del servicio nativo *$scope*, interaccionar con el Modelo; el cual

representa el estado real de la aplicación y proviene, en este proyecto, del servicio REST situado en el *back-end*.

Las aplicaciones construidas mediante SPA (del inglés, *Single-Page Application*) son aplicaciones de página única en las que todo el contenido HTML, CSS y JavaScript se carga de una vez. Esto provoca que, en aplicaciones relativamente grandes, tarde en cargar la primera vez que se visita, pero una vez dentro de la página, la navegación es mucho más fluida. Esto es gracias a que se hace uso de peticiones en segundo plano para agregar y modificar contenido dentro de la misma página; pero, en ningún caso recargándola. Y, aquí es donde AngularJS hace su función ocultando o mostrando contenido a través de la interacción con el usuario.

Gateway de servicios y patrón de micro-servicios.

En este apartado también se comentan los dos conceptos por su estrecha relación.

Tal como describe el Grupo de Trabajo de Ingeniería de Internet o IETF (del inglés, *Internet Engineering Task Force*): "*El Gateway o Puerta de enlace es un servidor que actúa como intermediario para otros servidores. A diferencia de un proxy, el Gateway recibe las peticiones como si fuera el servidor al cual va dirigido la petición para la obtención del recurso. El cliente que ejecuta la petición puede no ser consciente de la comunicación con el Gateway intermediario.*"

Y, en efecto, en el siguiente apartado se puede ver la estructura del sistema, en la que se observa que la única comunicación posible entre el cliente, mediante un navegador, y el sistema es a través del servidor que contiene una Gateway de servicios. Dicho servidor es el que hace de enrutador, mediante el servicio de Gateway, de las peticiones hacia el servidor (micro-servicio) correspondiente o, por lo contrario, procesa directamente la petición. Una vez procesada la petición por el micro-servicio, la respuesta es enviada de nuevo hacia la Gateway y ésta la reenvía al cliente que realizó la petición.

Tanto la aplicación que contiene la Gateway como cada micro-servicio son aplicaciones con estructura completa de Spring y *plugin* de Maven; con el correspondiente archivo pom.xml, donde se describen todas las librerías utilizadas en la aplicación.

El patrón de micro-servicios se basa en descomponer la aplicación en un conjunto de servicios colaboradores, donde cada servicio se encarga de implementar un conjunto reducido de funcionalidades relacionadas. Las ventajas de usar el patrón son las siguientes:

- La gran ventaja de los micro-servicios es que se desarrollan y se despliegan de forma individual, sin interferir unos con otros. Lo que permite organizar el

desarrollo con varios grupos de programadores y esto ayuda a la hora de generar nuevas versiones con alta productividad.
- Son relativamente reducidos en tamaño, lo que ayuda a su rápida comprensión por parte de un nuevo programador.

Los inconvenientes de usar el patrón también son variados:

- El desarrollo de un sistema con aplicaciones distribuidas añade complejidad al desarrollo, tales como:
    - Añade dificultad de testeo.
    - Necesidad de implementar mecanismos de comunicación entre servicios.
    - Implica alta comunicación entre los distintos equipos de programadores.
    - Se genera la dificultad en casos de uso que abarcan distintos servicios sin usar transacciones distribuidas.
- Más complejidad a la hora de desplegar el sistema.
- Se incrementa el uso de memoria necesaria, dado que cada servicio se convierte en una aplicación que se ejecuta en su propia JVM o máquina virtual de Java.

5.2 Estructura del sistema

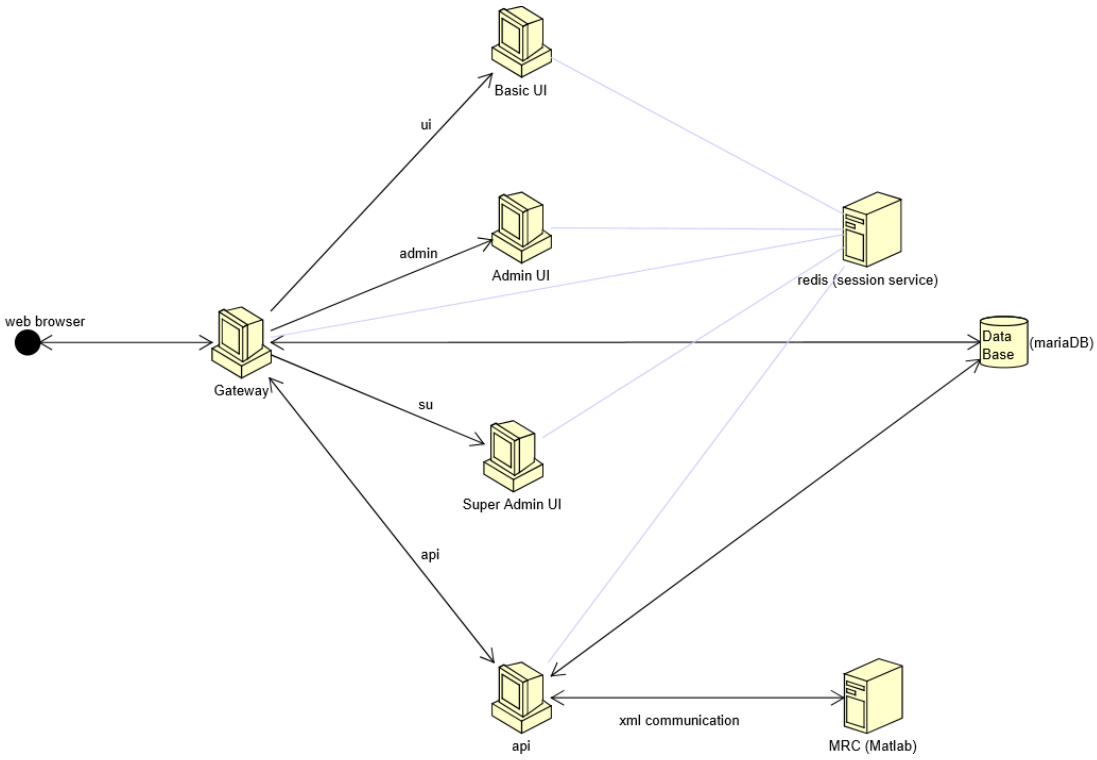


Ilustración 5-2

Como se puede observar en la ilustración superior, la estructura del sistema está formada por una puerta de enlace o Gateway como único punto de entrada al sistema y varios micro-servicios; donde cada uno es una aplicación java con estructura completa para ser ejecutada con el *framework* Spring a través de Maven. Además, el sistema mantiene la persistencia mediante una base de datos sobre MariaDB y, tanto la Gateway como todos los micro-servicios usan Redis para compartir las sesiones activas. Finalmente, y para cumplir con el objetivo del proyecto, se ha hecho uso del Matlab *Component Runtime* y poder, así, calcular las estructuras.

Como se ha mencionado anteriormente, existe la aplicación de entrada al sistema que implementa el patrón “Gateway” o puerta de enlace; y que es por donde pasan todas las peticiones del navegador que desean recibir servicio del sistema. Dicha aplicación está contenida dentro de un servidor que se mantiene a la escucha de recibir peticiones HTTP por el puerto 8080. Al recibir una petición, la aplicación decide si actuar como puerta de enlace, siempre que la petición tenga que ser redireccionada a un micro-servicio o, por lo contrario, tenga que ser directamente atendida por ella. La Gateway sólo atiende las peticiones de autenticación en el sistema y solicitud registro, ofreciendo al usuario los formularios correspondientes y haciendo uso de la

base de datos única para dichas acciones. Además, es en el único punto donde se puede crear una sesión, dentro del sistema, durante el proceso de autenticación del usuario; en los micro-servicios sólo se comprueba que exista la sesión del usuario correspondiente, pero en ningún caso la crean.

Así es como está configurado el servicio de Gateway dentro de la aplicación, mediante el servicio de Spring Cloud zuul:

```
zuul:
  routes:
    api:
      url: http://localhost:9000
      sensitive-headers:

    ui:
      url: http://localhost:8082
      sensitive-headers:

    admin:
      url: http://localhost:8081
      sensitive-headers:

    su:
      url: http://localhost:8083
      sensitive-headers:
```

Por detrás de la Gateway existen 4 aplicaciones que actúan como servicios colaboradores:

- El micro-servicio implementado en la aplicación “Basic UI” ofrece la interfaz de usuario con perfil: usuario básico. Está contenido dentro de un servidor que se mantiene a la escucha de recibir peticiones HTTP por el puerto 8082. Dado que no tiene acceso propio a la base de datos, extrae su información mediante peticiones HTTP Ajax – usando AngularJS – con la aplicación de tipo REST: “api”.
- El micro-servicio implementado en la aplicación “Admin UI” ofrece la interfaz de usuario con perfil: usuario administrador. Está contenido dentro de un servidor que se mantiene a la escucha de recibir peticiones HTTP por el puerto 8081. Y, dado que tampoco tiene acceso propio a la base de datos, extrae su información mediante peticiones HTTP Ajax – usando AngularJS – con la aplicación de tipo REST: “api”.
- El micro-servicio implementado en la aplicación “Super admin UI” ofrece la interfaz de usuario con perfil: usuario súper-administrador. Está contenido dentro de un servidor que se mantiene a la escucha de recibir peticiones

HTTP por el puerto 8083. E, igual que las dos anteriores interfaces de usuario, tampoco tiene acceso propio a la base de datos, por lo que extrae su información mediante peticiones HTTP Ajax – usando AngularJS – con la aplicación de tipo REST: “api”.

- El micro-servicio implementado en la aplicación “api” que ofrece el estado actual del sistema mediante el acceso completo a toda la información de la base de datos y al servicio principal de cálculo de estructuras desarrollado en Matlab. Está contenido dentro de un servidor que se mantiene a la escucha de recibir peticiones HTTP por el puerto 9000.

Finalmente se ha optado por utilizar el servicio externo de mantenimiento de sesión compartida por todas las aplicaciones; ofrecido por el servicio de entorno Linux “REDIS” y adaptado a todos los micro-servicios como *plugin* del *framework* Spring. Asimismo, la intención es que en un futuro se implemente el protocolo de autenticación OAUTH2 con varios servidores de identidades ofrecidos por las principales redes sociales tales como: Google, Facebook, LinkedIn, Twitter, etc. Y, en consecuencia, se elimine la necesidad de utilizar el servicio REDIS y el sistema pueda ser distribuido completamente (con los correspondientes cambios en la base de datos y algunos nuevos procesos de comunicación entre servicios).

En el apartado correspondiente se detalla, con la ayuda de diagramas de secuencia, cómo interaccionan los distintos elementos del sistema para los casos de uso más relevantes.

5.3 Base de datos

Dado que el diagrama de base de datos completa es bastante extenso (ver diagramad de base de datos completo en anexo 1), se ha separado el diagrama en tres partes: La parte de usuario, la parte de datos generales de la estructura con los datos de entrada y, una tercera parte con los datos de salida de la estructura.

En la siguiente ilustración se observa la primera parte de datos de usuario:

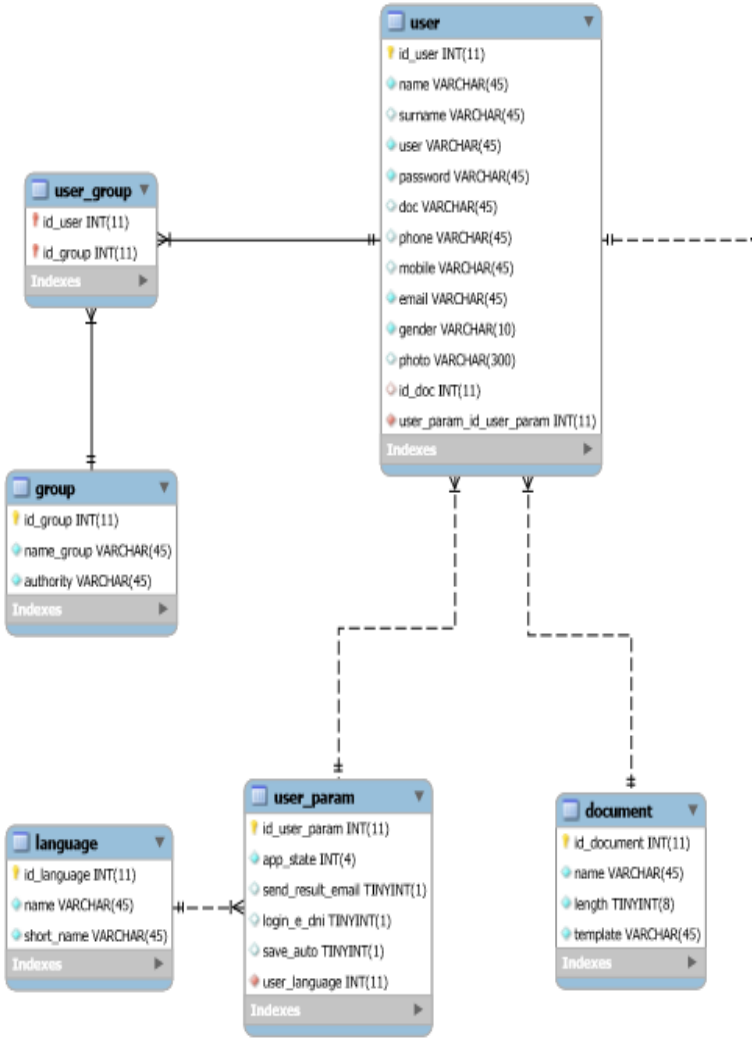


Ilustración 5-3

En la ilustración superior se puede observar la parte del diagrama la base de datos correspondiente al usuario; donde se pueden apreciar las siguientes tablas:

- *user*: es la tabla principal de usuario y es de donde cuelgan todas las demás correspondientes al usuario. En ella se almacenan todos sus datos básicos y, además, es la tabla que relaciona el usuario con la tabla de estructuras, que se encuentra en la siguiente parte del diagrama.
- *group*: es la tabla que almacena la información de los grupos de usuario que existen en la aplicación; contiene el campo *authority* que contiene qué tipo de usuario es: básico, administrador o súper usuario.
- *user_group*: es una tabla de relación entre la tabla *user* y la tabla *group*, ya que mantienen una relación de n a n; un usuario puede estar en varios grupos y un grupo puede tener varios usuarios.
- *user_param*: esta tabla contiene todas las opciones configurables para un usuario, como el estado del usuario respecto a la aplicación; almacenado en el campo *app_state* y que da la posibilidad de implementar el caso de uso de solicitud de registro, o de inhabilitar al usuario, entre otros. La relación con la tabla *user* es de uno a uno, ya que un usuario sólo puede tener una configuración asociada.
- *language*: esta tabla contiene todos los idiomas configurados para ofrecer servicios personalizados respecto al idioma. La relación con *user_param* es de uno a n, ya que un usuario sólo puede tener un idioma preferente.
- *document*: esta tabla almacena todos los tipos de documentos que el usuario puede presentar a la aplicación como DNI, NIE, etc. Actualmente, no se utiliza, pero está pensada para dar una funcionalidad de ayuda a la hora de rellenar los distintos tipos de documentos y proporcionarla en caso de errores en los campos. Para ello se deberán rellenar los campos *length* y *template*.

En la siguiente parte del diagrama de base de datos se encuentran: la tabla donde se almacena la información general de la estructura y la tabla donde se almacena la información de entrada a la rutina de Matlab.

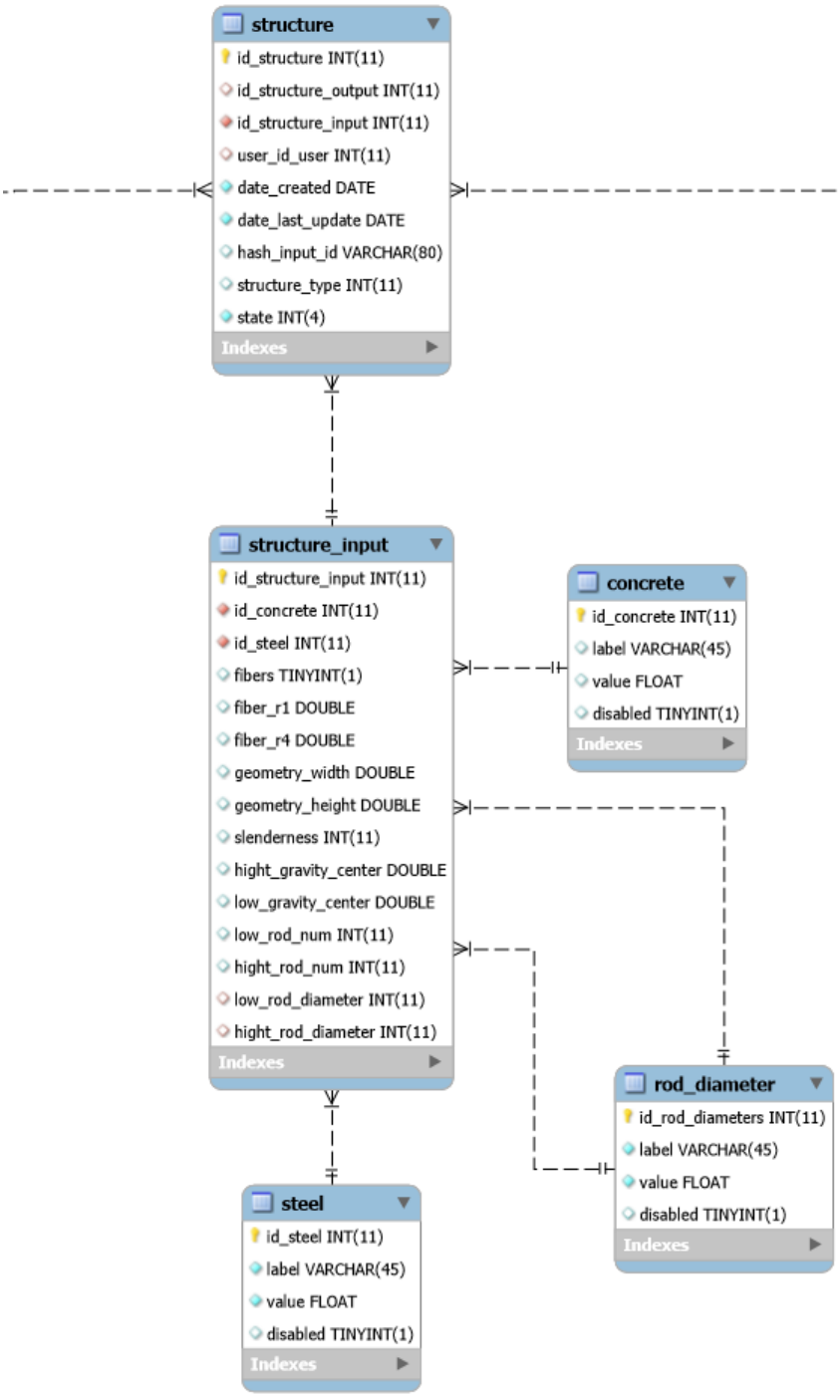


Ilustración 5-4

En la ilustración superior que muestra las partes comentadas de la estructura, se pueden observar las tablas siguientes:

- *structure*: Es la tabla donde se almacenan los datos generales de la estructura y de donde cuelgan todas las demás tablas relacionadas con el cálculo de estructura. Además, es la tabla con la cual se relaciona la tabla principal de usuario, *user*, en la que la relación es de uno a n, dado que un usuario puede tener varios cálculos de estructura hechos, pero un cálculo de estructura sólo puede tener un usuario propietario.
- *structure_input*: es una de las dos tablas que cuelgan directamente de la tabla *structure* y es donde se almacenan todos los datos relacionados con la entrada de datos, por parte del usuario, para el cálculo.
- *concrete, steel* y *rod_diameter*: son las tres tablas relacionadas con la recién expuesta *structure_input*, con relación de tipo uno a uno, donde se almacenan los distintos valores, seleccionables por parte del usuario básico y gestionables por parte del súper usuario, para cumplir con la normativa de valores en el cálculo de la estructura.

En la siguiente ilustración se muestra la tercera parte del diagrama completo de la base de datos, y, a la vez, la segunda parte del diagrama de la parte de la estructura, correspondiente a los datos de salida de la rutina de Matlab.

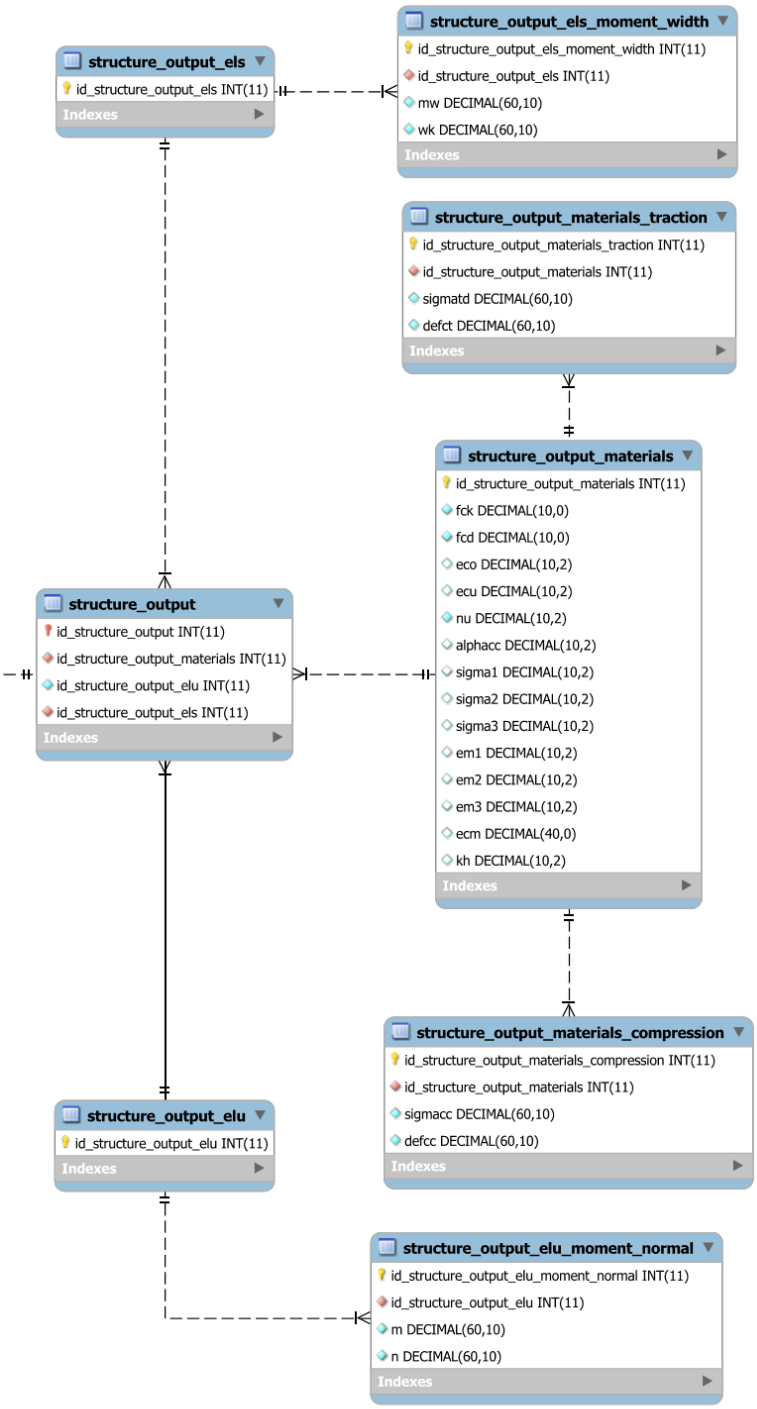


Ilustración 5-5

Como se puede observar en la ilustración superior, donde se muestra la parte de salida de la estructura, se ha diseñado esta parte de la base de datos para almacenar los resultados que devuelve la rutina de Matlab. Sus tablas se detallan a continuación:

- *structure_output*: Es la tabla principal de esta segunda parte del diagrama correspondiente a la estructura. Esta es una tabla que no contiene información propiamente de datos de salida, sino que contiene las relaciones necesarias de tablas. Está directamente relacionada con *structure* y con relación de tipo uno a uno.
- *structure_output_els*: Esta tabla está preparada para almacenar toda la información relacionada con el concepto ELS de salida de la estructura. Está relacionada con *structure_output* con relación de tipo uno a uno.
- *structure_output_els_moment_width*: Esta tabla almacena la información de la gráfica "Momento-Anchura de fisura" correspondiente al ambiente del ELS Fisuración de salida de la estructura. Tiene una relación de tipo uno a n con la tabla *structure_output_els*, ya que se trata de la información de una gráfica de dos dimensiones en asociada al concepto ELS Fisuración.
- *structure_output_elu*: Esta tabla hace el mismo papel que la tabla *structure_output_els* pero dentro del ambiente del concepto ELU Flexión de salida de la estructura.
- *structure_output_elu_moment_normal*: Esta tabla almacena la información de la gráfica "Momento-Normal" correspondiente al ambiente del ELU Flexión de salida de la estructura. Tiene una relación de tipo uno a n con la tabla *structure_output_elu*, ya que se trata de la información de una gráfica de dos dimensiones en asociada al concepto ELU Flexión.
- *structure_output_materials*: Esta tabla almacena toda la información relacionada con el concepto materiales de salida de la estructura. Está relacionada con *structure_output* con relación de tipo uno a uno.
- *structure_output_materials_compresion*: Esta tabla almacena la información de la gráfica "Compresión" correspondiente al ambiente de los materiales de salida de la estructura. Tiene una relación de tipo uno a n con la tabla *structure_output_materials*, ya que se trata de la información de una gráfica de dos dimensiones en asociada al concepto materiales.
- *structure_output_materials_traction*: Esta tabla es idéntica a la anterior tanto en estructura y como en relaciones pero almacena la información de la gráfica "Tracción".

Y esta es toda la estructura de la base de datos del sistema. En el siguiente apartado se ve cómo interacciona el usuario con el sistema y sus distintas partes.

5.4 Diagramas de secuencia

Los diagramas de secuencia ayudan a visualizar las interacciones entre las distintas partes del sistema, sin los cuales sería más dificultoso entender algunos casos de uso.

Cálculo de la estructura

Con intención de ilustrar el proceso completo del principal objetivo, la siguiente ilustración presenta un diagrama de secuencia, con la interacción de los distintos componentes, para el cálculo de la estructura; desde que el usuario básico presiona el botón de "Nueva estructura", que se encuentra en el listado de estructuras de su interface gráfica, hasta que recibe el correo electrónico conforme el resultado ya está disponible. (Anexo 2 está la imagen ampliada)

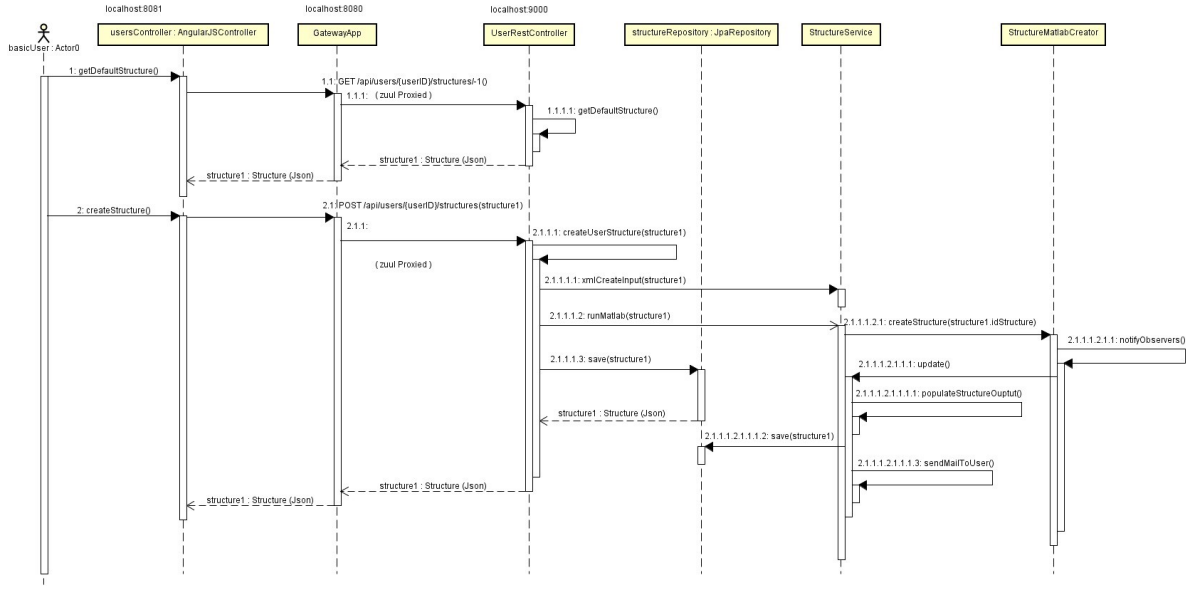


Ilustración 5-6

Como se puede observar, el proceso de cálculo de la estructura se divide en dos acciones llevadas a cabo por parte del actor usuario básico, las cuales se describen a continuación:

Presión del botón "Nueva estructura": Esta acción genera que un evento JavaScript llame al método *getDefaultStructure()*; el cual se encuentra dentro del controlador de Angular *UsersController*. Dicho método usa el servicio creado *users*, el cual, a su vez, usa el servicio nativo de Angular, *http*, para generar una petición HTTP, de tipo GET, al recurso descrito por la URL relativa "*/api/users/{userID}/structures/{structureID}*", donde *{structureID}* tiene el valor -1. En

este caso, la petición va sin parámetros ni cuerpo e, igual que todas las peticiones, ésta es dirigida y procesada, en primera instancia, por la aplicación "Gateway", que redirige la petición al micro-servicio "api", con URL: http://localhost:9000/.

Una vez en el micro-servicio "api", y después de pasar por su *dispatcher servlet* de Spring, esta petición es procesada por el *handler UserRestController*, ya que todo el controlador está mapeado con el *path /users*.

Asimismo, dentro del *handler* hay varios métodos mapeados con el resto del *path* de la petición y filtrados por métodos HTTP, de tal manera que el método *readUserStructure(@PathVariable Integer userId, @PathVariable Integer structureId)* es el que filtra más específicamente la petición y, por lo tanto, es el que la procesa:

- Asigna el parámetro *{structureID}*, en este caso con valor -1, a su correspondiente argumento.
- Crea un objeto de clase *Structure* a partir de las clases del dominio, con todas las relaciones correspondientes y valores por defecto.
- Devuelve la estructura, en formato JSON y que acaba de generar, hacia la "Gateway".

La aplicación "*Gateway"* sólo ha hecho de enrutador en la petición, con lo que coge la respuesta HTTP y la devuelve tal como le ha sido entregada, por parte del micro-servicio "api", hacia el cliente, es decir, hacia el controlador de Angular *UsersController*.

El controlador *UsersController* genera un objeto estructura con los datos recibidos en la respuesta HTTP, en formato JSON, de manera automática usando los inicializadores de objetos de JavaScript. Dicha estructura de datos es la que se usa a la hora de inicializar el contenido del formulario del cálculo de estructura.

Una vez realizado el proceso descrito anteriormente, ya se está en disposición de asignar valores a los campos del formulario correspondientes a los datos de entrada de la estructura. Una vez rellenados correctamente, presionar el botón de cálculo de estructura, que se detalla a continuación.

Presión del botón "Calcular estructura": El hecho de presionar sobre dicho botón genera un evento de JavaScript que llama al método *createStructure()* del controlador de Angular *UsersController*. Este método usa el servicio creado *users*, que a su vez usa el servicio nativo de Angular *http* para generar una petición HTTP, de tipo POST, a la URL "*/api/users/{userID}/structures/*". Dicha petición, como se puede observar en la URL, no contiene parámetros pero sí que contiene cuerpo; el cual está formado por el objeto, en formato JSON, de tipo estructura y con los valores correspondientes a los parámetros de entrada de la estructura modificados por el usuario. Esta petición, como todas las demás, ésta es dirigida y procesada, en primera

instancia, por la aplicación “Gateway”, y, en este caso la redirige al micro-servicio “api”, con URL: http://localhost:9000/.

Ya siendo procesada por la aplicación que ofrece el servicio “api“, después de procesarla el *dispatcher servlet* de Spring es delegada al *handler UserRestController* y, el método *createUserStructure(structure1)* es el quien la procesa; siguiendo los pasos que se describen a continuación:

1. Con la ayuda del servicio Jackson la estructura en formato JSON es transformada a un objeto de dominio de tipo *Structure*, con todas las relaciones y valores.
2. Usa el método *xmlCreateInput*, pasando como argumento la estructura recibida y transformada a objeto de dominio de clase *Structure*, del servicio creado *StructureService*. Este método genera el fichero en formato XML (del inglés, *eXtensible Markup Language*) con todos los parámetros necesarios para el cálculo de la estructura, por el servicio externo MCR.

   Una vez termina el método anterior, se ejecuta un nuevo hilo de ejecución, en paralelo, para gestionar el proceso externo que calcula la estructura, porque dicho proceso dura más de 10 segundos y no se puede hacer esperar tanto al cliente hasta recibir una contestación por parte del servidor web. Además, cuantas más peticiones haya en cola del MCR, mayor tiempo de espera; por la secuencialidad del MCR a la hora de gestionar las peticiones que procesa.

3. En el hilo principal, simplemente, se guarda la estructura en la base de datos usando el objeto *structureRepository* y se devuelve, en formato JSON, a la “Gateway”, que la redirige al cliente.

   Para la creación y gestión del hilo nuevo creado, se usa el patrón de software, *Observer* (véase su referencia); donde la clase *StructureService* implementa la *interface Observer* y la clase *StructureMatlabCreator* extiende la clase *Observable* de tal manera que se pueda implementar el patrón con objetos de las mismas.

4. Como se puede observar en el diagrama, en el hilo principal, justo antes de guardar la estructura y justo después de usar el servicio *StructuresService* para generar el fichero XML llamando al método *xmlCreateIntput*, se utiliza de nuevo el mismo servicio para llamar al método *runMatlab*, y éste realiza las siguientes tareas:

   - Instancia un servicio de creación de hilos y se pone a la escucha o, lo que es lo mismo, se mantiene observando hasta que el servicio

instanciado avise a sus observadores de la finalización del nuevo hilo creado.

- Una vez el servicio externo del MCR finaliza la ejecución de la rutina de Matlab, leyendo las variables de entrada del fichero XML creado, el servicio de creación de hilos notifica que el proceso ha terminado, llamando al método heredado *notifyObservers*, el cual genera un evento que recibe su observador y ejecuta su método *update*, el cual realiza las tareas de servicio del post-cálculo correspondientes:
    - Recoge los resultados que el proceso de Matlab ha depositado en un fichero en formato XML
    - Rellena todos los valores y relaciones correspondientes a la parte de salida de la estructura del objeto structure1, es decir; crea el objeto de clase *StructureOutput* y todas sus relaciones y los rellena con los valores que están dentro del fichero XML de salida de la rutina de Matlab
    - Vuelve a guardar la estructura completa en base de datos mediante el método *save* del repositorio *structureRepository*
    - Finalmente, envía un correo electrónico al usuario que ha realizado la petición de cálculo mediante la llamada al método *sendMailToUser*.

6. Manual de usuario

En este manual de usuario se detallan los pasos a realizar para llevar a cabo los casos de uso principales de la aplicación.

Entrar a la aplicación

Para entrar en la aplicación hay que rellenar correctamente el formulario que hay en la página de inicio a la aplicación, como se puede observar en siguiente imagen:

Ilustración 6-1

Solicitud de registro

Donde se puede observar que, desde la página inicial, se puede acceder al recurso de solicitud de registro, el formulario para el cual se muestra en la siguiente imagen:

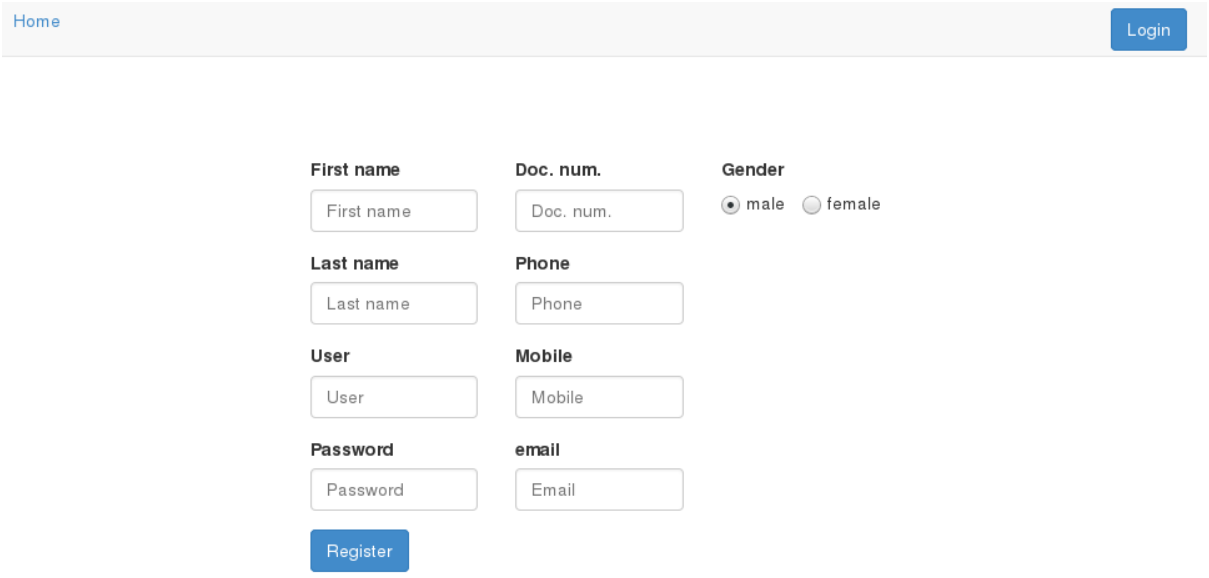


Ilustración 6-2

Una vez rellenado correctamente el formulario de entrada a la aplicación, que se ve en la ilustración 6-1, y presionado el botón de “Log in” se entra a la aplicación y se muestra el siguiente menú:

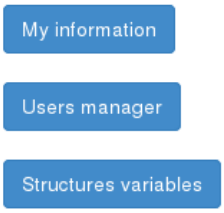


Ilustración 6-3

Donde los tres menús se muestran porque el usuario actual tiene los tres perfiles de usuario de la aplicación: Usuario básico, Usuario administrador y súper-usuario.

El bloque de información propia de usuario “My información”, donde se encuentra el perfil de usuario, su configuración y su listado de estructuras calculadas, es bloque perteneciente al perfil de usuario básico. El bloque de gestión de usuarios “Users manager”, donde se encuentra el listado con todos los usuarios, es el bloque perteneciente al perfil de usuario administrador. El bloque de gestión de las variables de las estructuras, donde se encuentran los formularios de gestión de las variables de entrada predefinidas para el cálculo de estructuras, corresponde al último perfil de usuario: súper-usuario.

Presionando el botón de “*My information*” se accede a la siguiente página:

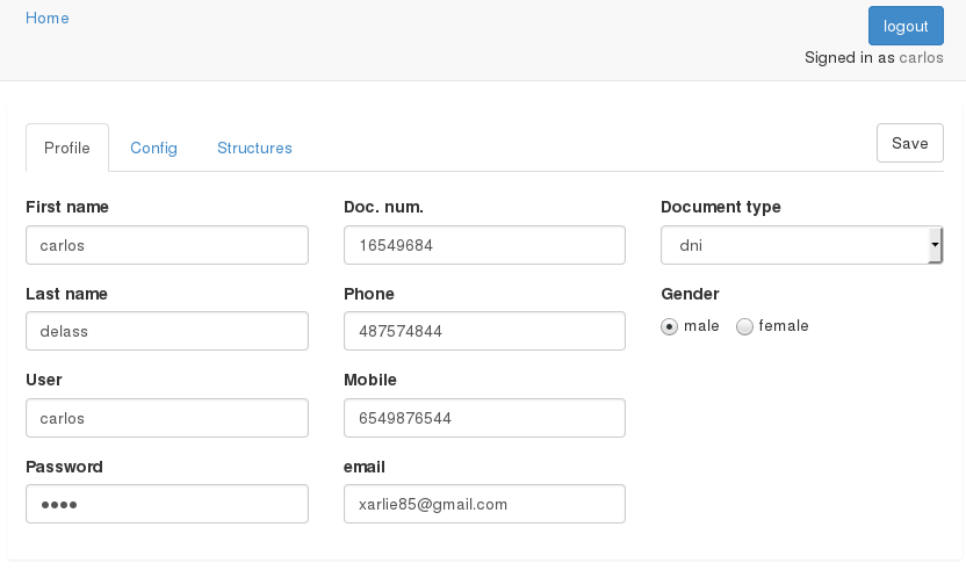


Ilustración 6-4

Que, como se puede ver, es el perfil de usuario del usuario conectado a la aplicación: “carlos”. También, se puede observar que hay dos pestañas más; una de configuración de variables del usuario, la cual se ilustra en la siguiente imagen:

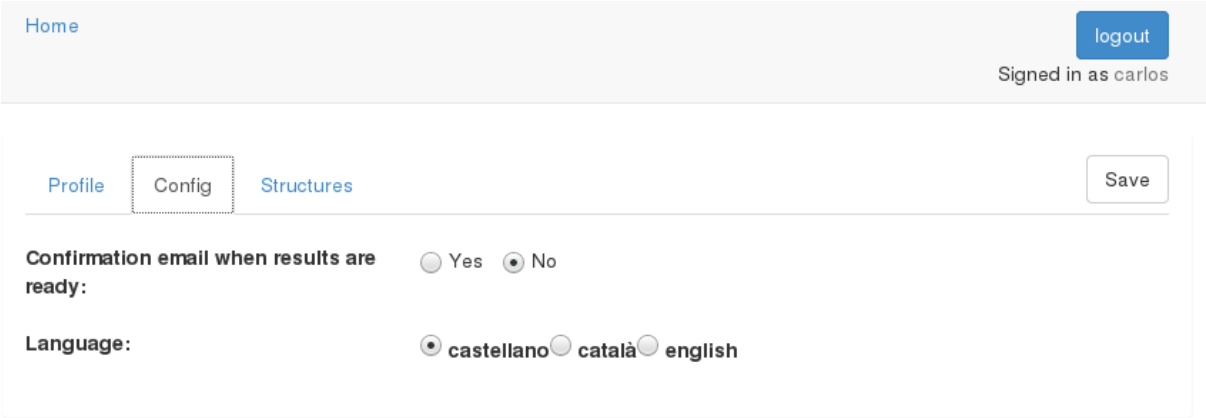


Ilustración 6-5

Y, otra donde se ubican todas las estructuras calculadas por el usuario conectado a la aplicación:

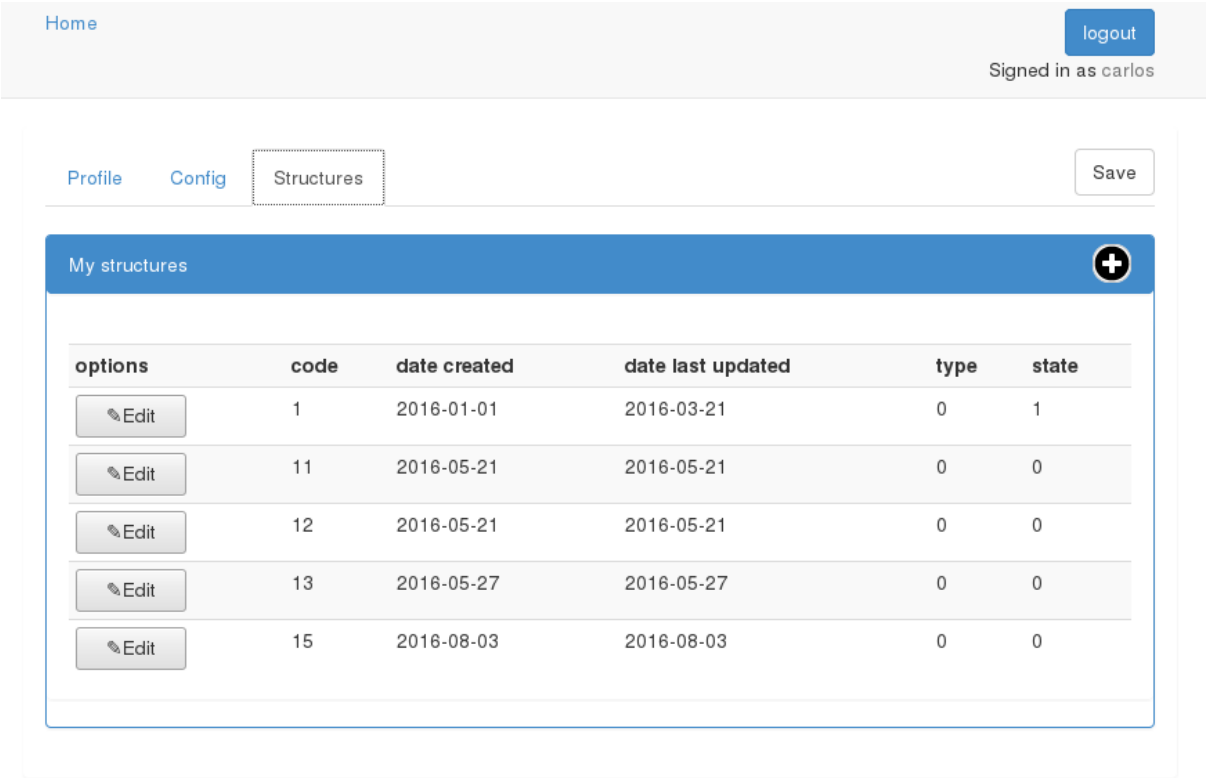


Ilustración 6-6

Cualquier cambio que se haga en las dos primeras pestañas, correspondientes a las ilustraciones 6-4 y 6-5, deberá presionarse el botón "*Save*":

Ilustración 6-7

En la tercera pestaña, para la gestión de sus propias estructuras calculadas, si el usuario desea crear una estructura nueva deberá presionar el botón con el signo "+", siguiente:

Ilustración 6-8

En caso de que el usuario quiera gestionar o recalcular cualquiera de las estructuras ya calculadas deberá presionar el botón de editar siguiente:

Ilustración 6-9

Cálculo de estructura

Y, cualquiera de las dos opciones llevará a mostrar el formulario de estructura compartido por las dos opciones de creación y edición, que se muestra en la siguiente imagen:

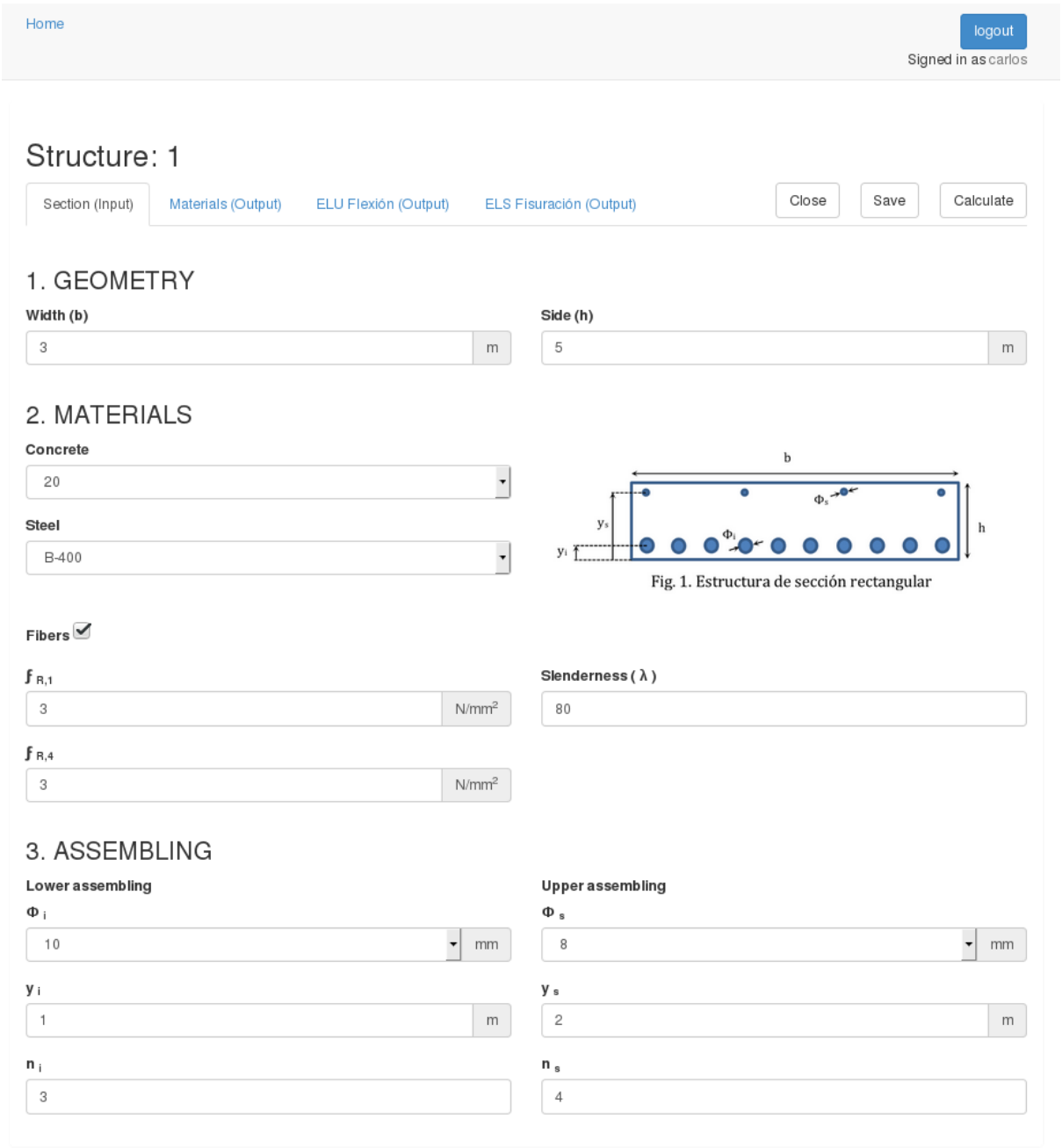


Ilustración 6-10

Una vez rellenadas todas las variables de entrada, hay tres opciones:

- Presionar el botón "*Close*": Cierra la estructura y vuelve a la vista de la ilustración 6-6.

Ilustración 6-11

- Presionar el botón "*Save*": Guarda la estructura con los valores actuales, tanto si han sido modificados como si no lo han sido, almacenados en el objeto de tipo estructura del código Angular situado en la parte del cliente.

Ilustración 6-12

- Presionar el botón "*Calculate*": Este botón, de la siguiente figura, es el que genera el cálculo de la estructura en el servidor y hace que se rellenen las otras pestañas de *output* con los resultados generados en el servidor.

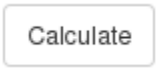


Ilustración 6-13

Las pestañas de *output*, que pertenecen a los resultados de la aplicación de Matlab son los que se muestran a continuación:

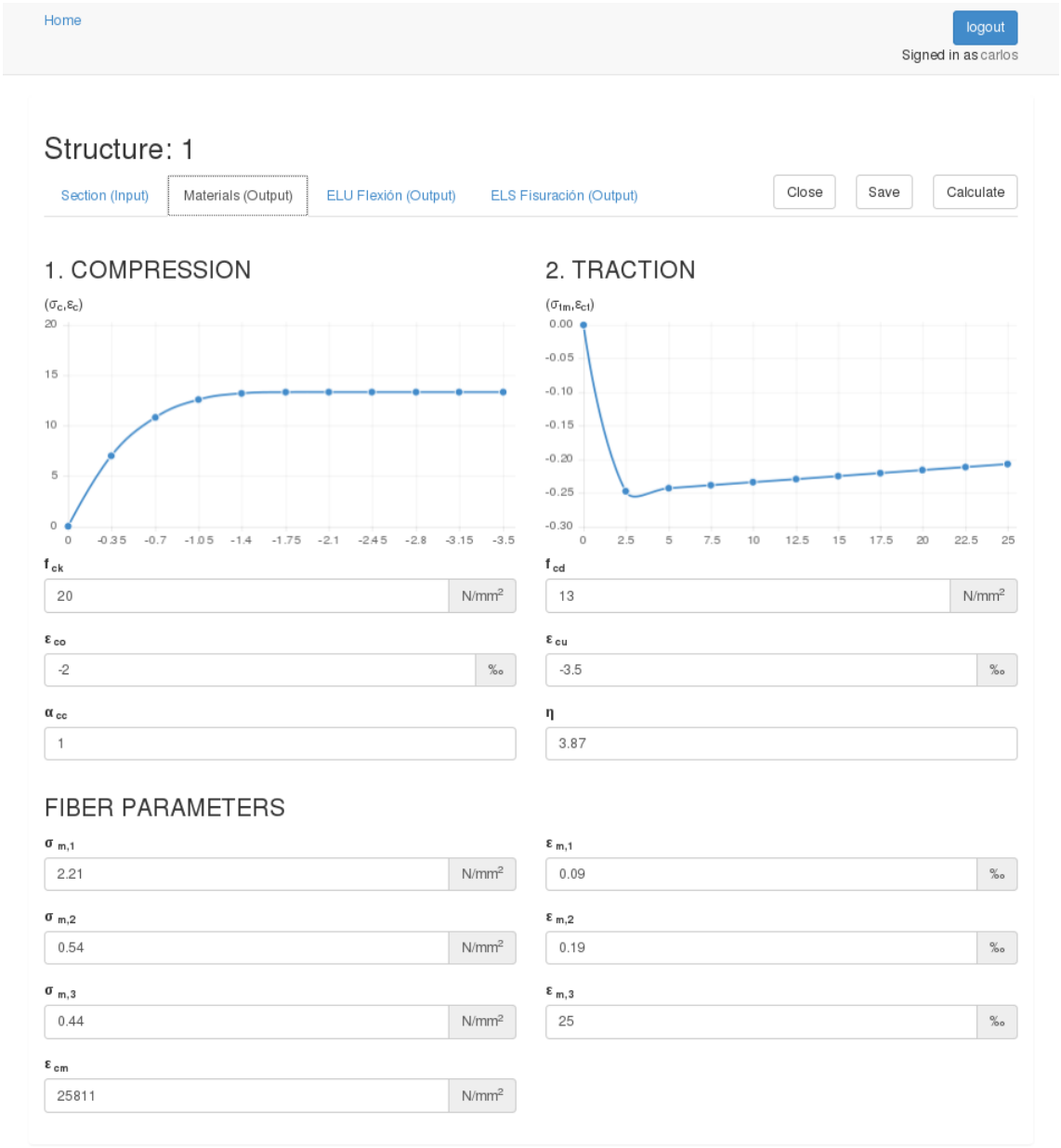


Ilustración 6-14

Donde se puede observar que la ilustración 6-14 muestra los resultados del cálculo de la estructura correspondientes a la parte de los materiales, como segunda pestaña de la interface gráfica. En la siguiente imagen se muestra la segunda parte de los resultados del cálculo, correspondientes a la parte de ELU Flexión, y tercera pestaña:

Home

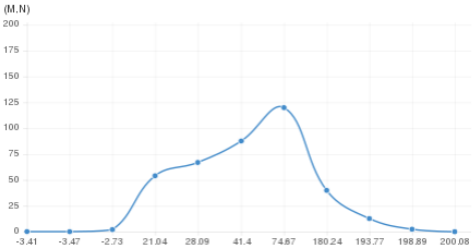
logout

Signed in as carlos

Structure: 1

Section (Input) | Materials (Output) | ELU Flexión (Output) | ELS Fisuración (Output) | Close | Save | Calculate

1. MOMENTUM - NORMAL



Values table

Momentum (KNm)	Normal (KN)
157.201927	-3414.542557
144.050538	-3424.011132
130.899148	-3433.479706
117.747759	-3442.948281
104.596369	-3452.416855
91.44498	-3461.88543
78.29359	-3471.354004
65.142201	-3480.822578
51.990811	-3490.291153
38.839422	-3499.759727
25.688032	-3509.228302
12.536642	-3518.696876
2057.261148	-2733.378485
17178.032573	3592.109045
32084.510842	10231.954074
45106.375439	16477.492105
50860.450369	19395.432546
52569.08127	20268.392851
54078.019052	21041.079736
56039.756478	22070.886724
57867.444929	23037.953787
59819.436437	24084.095374
62218.670112	25407.393861
64522.894867	26693.714256
66979.917691	28091.609902
69686.277274	29668.930492
72695.234886	31473.850955
76025.353633	33538.15272
79655.160079	35871.354232
83396.34658	38361.107343
87711.757111	41396.641477
92269.138711	44784.488667
97224.785714	48740.253183
102586.139333	53431.131784
108326.357223	59103.607665
114236.200069	66012.91315
119945.20499	74668.457731
124560.634041	85890.934697
125838.844246	100887.748829
118228.800263	121968.150348
86029.539474	153739.802735
46991.201897	176754.66885
39979.789712	180235.040611
33774.291925	183315.524621
28311.730911	186027.399833
23531.52268	188400.637498
19375.490903	190464.194207
15787.88168	192245.704563
12715.379158	193771.623431
10107.122075	195067.187729
7914.721354	196156.407699
6092.278863	197062.057595
4596.407521	197805.665714
3386.252912	198407.503861
2423.516673	198886.574754
1672.618032	199260.329067
1100.66358	199544.76582
676.835582	199755.703508
373.340893	199906.93285
165.09934	200010.89841
29.787918	200078.676869

Ilustración 6-15

Y finalmente, la tercera parte de los resultados del cálculo de la estructura, que se ubican en la cuarta y última pestaña de la interface gráfica de ésta, se muestra a continuación:

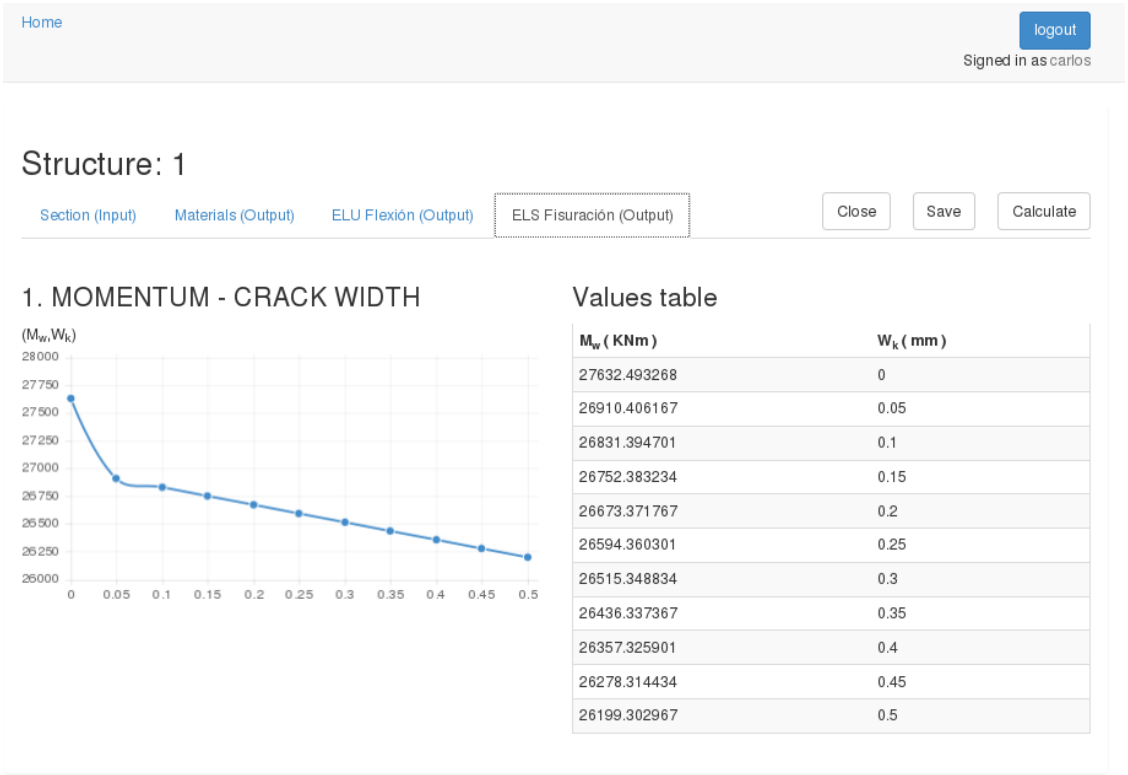


Ilustración 6-16

Y corresponde a la parte de ELS Fisuración de la salida del cálculo.

Hasta aquí se ha detallado la parte de la interfaz gráfica correspondiente al perfil de usuario básico, es decir la interface gráfica de usuario con permisos básicos. Si retrocedemos hasta el menú de la ilustración 6-2, veremos que el siguiente apartado a comentar es el apartado al que nos dirige la acción de presionar el botón "*Users Manager*", de la siguiente imagen, el cual sólo se muestra y se puede acceder si el usuario conectado posee permisos de administrador.

Ilustración 6-17

Gestión de usuarios

En la siguiente ilustración, se muestra la página de inicio de la gestión de usuarios:

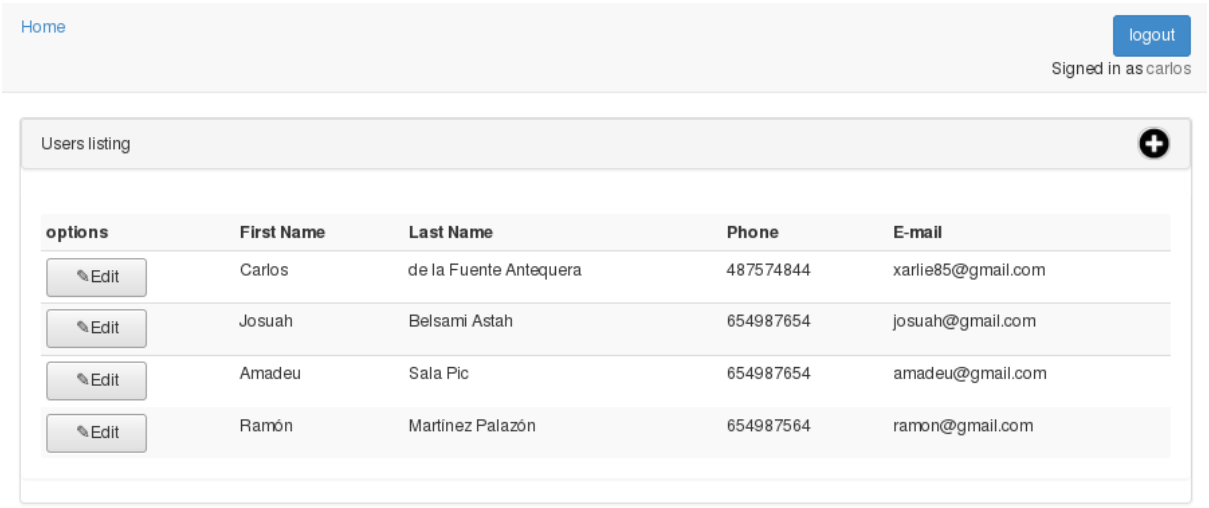


Ilustración 6-18

Donde se puede observar una lista de usuarios, en formato de tabla, que se pueden editar presionando el botón de edición situado en cada fila, como el de la imagen siguiente:

Ilustración 6-19

Y, de igual manera, presionando el botón de añadir usuarios, como el de la siguiente imagen:

Ilustración 6-20

Ambos, llevarán a mostrar el formulario que comparten las dos acciones de inserción/modificación de usuarios, que se muestra en la siguiente ilustración:

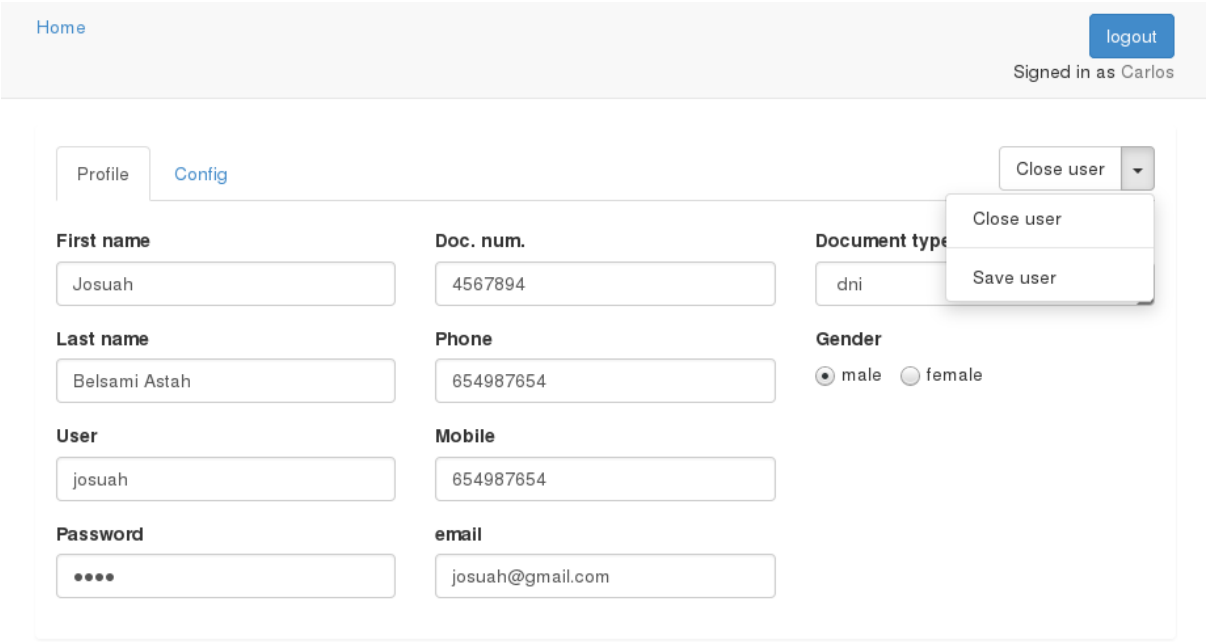


Ilustración 6-21

En el cual se pueden observar dos pestañas: La actual de perfil de usuario, en la que se encuentran todos los campos personales a rellenar para la modificación/inserción de un usuario; y, la que se muestra en la siguiente imagen de parámetros de configuración:

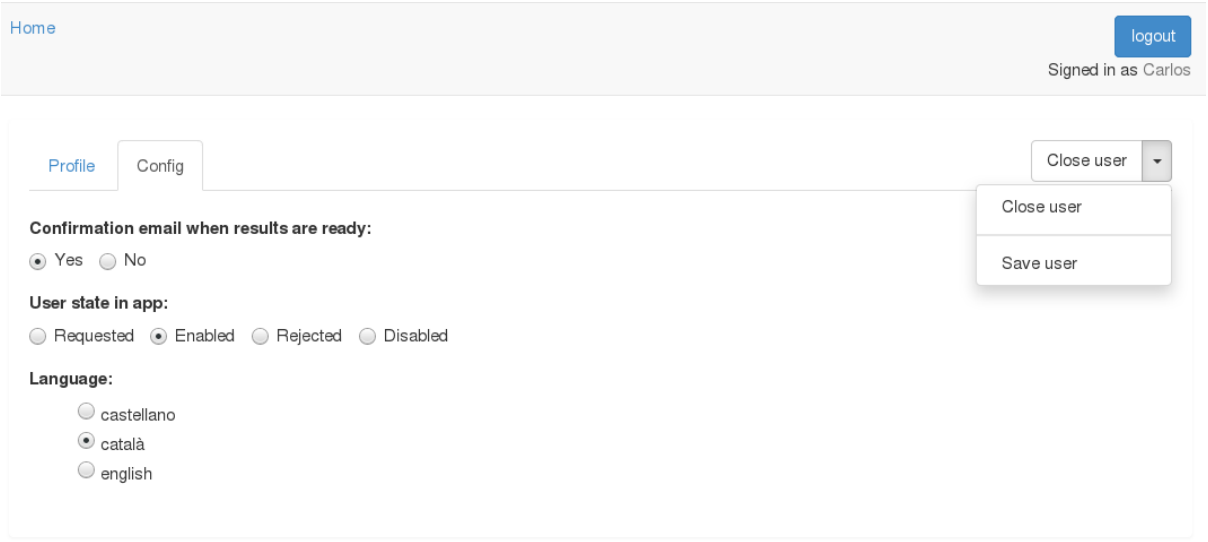


Ilustración 6-22

Donde hay los parámetros de configuración necesarios para el administrador, como es el caso del estado del usuario en la aplicación: Con éste, únicamente el

administrador tiene la capacidad de deshabilitar al usuario, o si se trata de un usuario de internet que ha solicitado el registro en la aplicación, también el administrador va a poder aceptarlo (*Enabled*) o rechazarlo (*Rejected*). Finalmente, en caso de que el administrador quiera grabar los datos modificados, tendrá que usar la opción "*Save user*" del selector que se muestra, también, en la siguiente imagen.

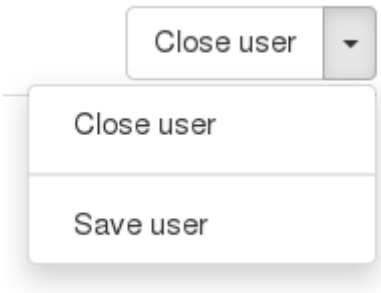


Ilustración 6-23

Y, aquí finaliza la parte del manual correspondiente a la gestión de usuarios del perfil administrador. Seguidamente, se detallan los pasos que hay que seguir para la gestión de variables de entrada de la estructura, las cuales únicamente las puede gestionar el usuario que contenga permisos de súper-usuario.

Hay que volver a recordar el menú de la ilustración 6-2, en el que hay el menú de la siguiente imagen, oportuno para llegar a la interface gráfica del súper-usuario:

Ilustración 6-24

Presionar dicha opción hace que aparezca la siguiente interface gráfica:

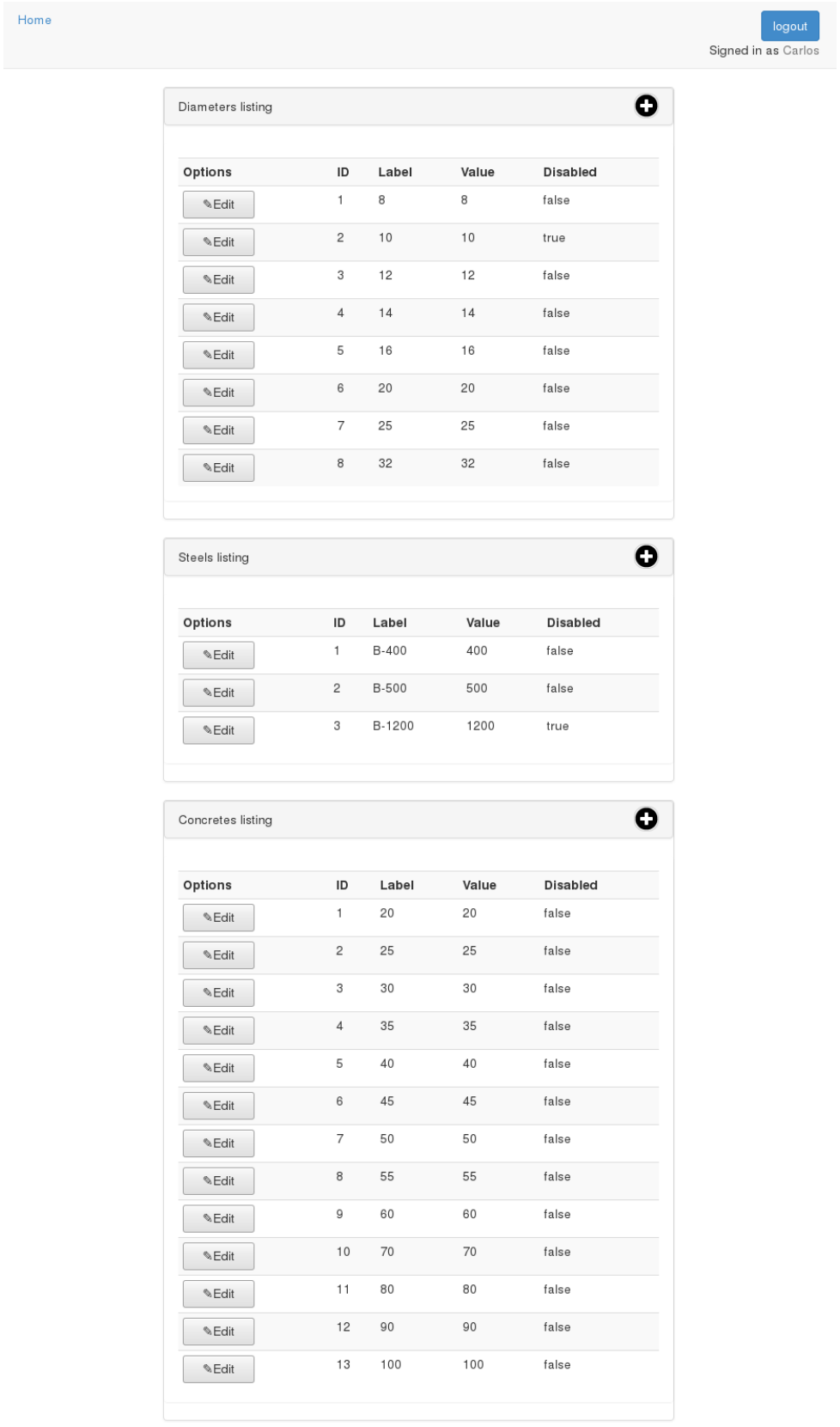


Ilustración 6-25

Donde están los listados de los valores pertenecientes a cada una de las variables de entrada para el cálculo de la estructura. Entonces, es aquí donde el súper-usuario puede gestionar los estándares para los tres tipos de variables: Diámetros, Aceros y Hormigón.

Dado que los tres tipos de variables se gestionan en la misma interface y de la misma manera, se comentan, en general, las acciones que se pueden ejecutar.

Para editar cualquier registro, se debe presionar su respectivo botón de editar:

Ilustración 6-26

Y, en caso de querer añadir un registro con valor nuevo, se debe presionar el botón de añadir de la tabla deseada:

Ilustración 6-27

Cualquiera de las dos opciones hará que aparezca una nueva fila encima de la tabla, tal como muestra la siguiente imagen; con los campos de los datos rellenados en el caso de estar modificando el registro, como sin datos para el caso de un registro nuevo:

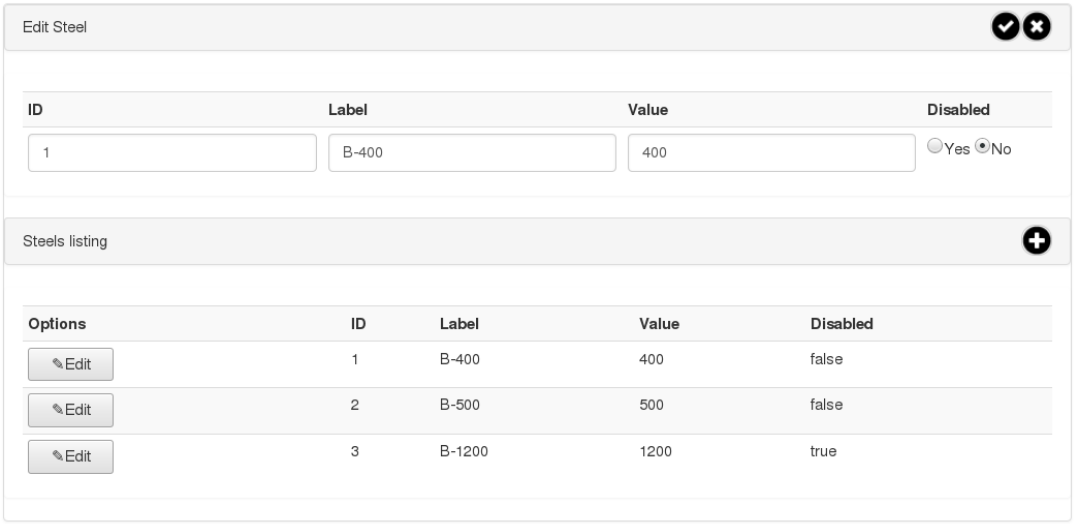


Ilustración 6-28

Donde se ha presionado el registro con ID igual a 1 de la tabla de Aceros y, por ello, aparecen sus datos en el registro de modificación.

Igualmente, si añadimos un registro aparecerá de esta siguiente forma, con los campos por rellenar y, lo más importante, sin campo ID; ya que se rellenará de forma automática durante la petición:

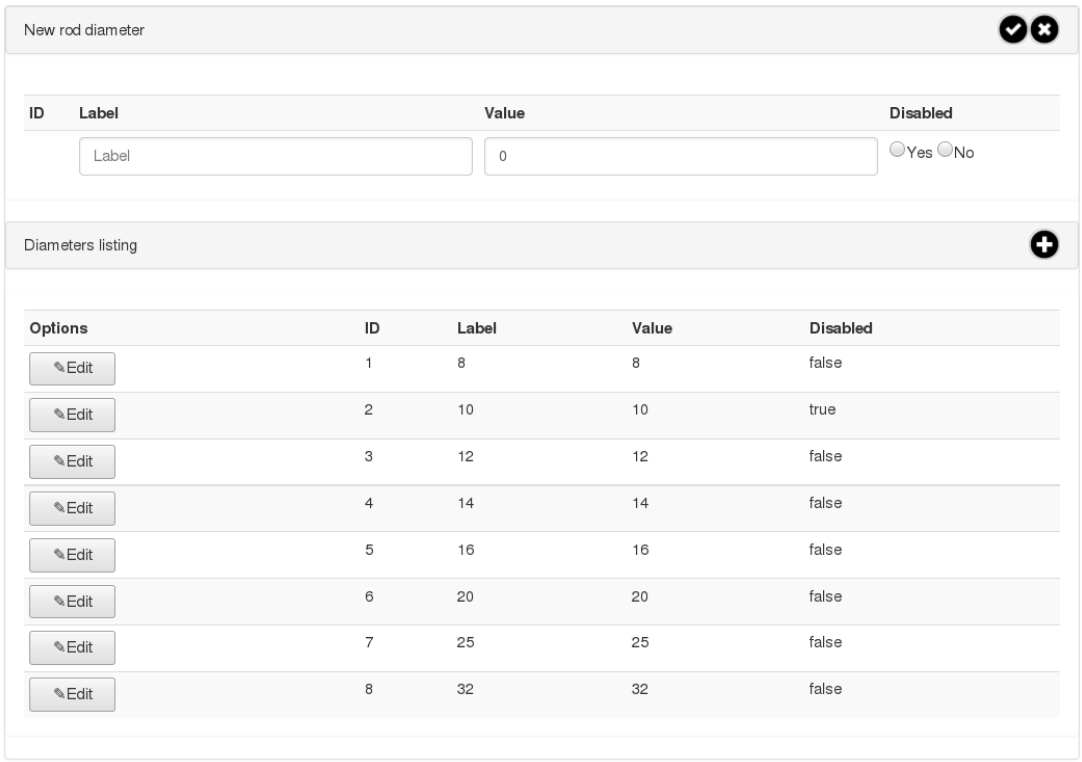


Ilustración 6-29

Y, ahora sí que, una vez acabado el manual de usuario correspondiente al perfil de súper-usuario, ya se han comentado todas las posibles acciones que un usuario cualquiera puede realizar.

7. Manual del programador

Para desarrollar este sistema se han llevado a cabo una serie de pasos que es necesario detallar para futuras ampliaciones y para poder comprobar el funcionamiento del sistema.

Con sistema operativo Windows: Dado que el sistema hace uso de algunos servicios propios de entorno Linux, será necesario, por el momento, instalar una máquina virtual con sistema operativo Linux, como huésped.

El gestor de máquinas virtuales utilizado en el proyecto es *Oracle VM VirtualBox*, descargable desde https://www.virtualbox.org/.

Una vez descargado e instalado, hay que crear una máquina virtual con el sistema operativo Linux: El sistema operativo utilizado para el proyecto ha sido Fedora 22 (4.4.8-200.fc22.x86_64). Y, da paso a poder seguir como si trabajásemos directamente con un sistema operativo Linux, como describe el siguiente apartado.

Con sistema operativo Linux: Con este sistema operativo se puede empezar directamente instalando los componentes necesarios para ejecutar el proyecto y que se listan a continuación, junto a sus versiones.

- MariaDB: *Ver 15.1 Distrib 10.0.23-MariaDB, for Linux (x86_64) using readline 5.1*
- Docker: docker-1.8.2-7.gitcb216be.fc22.x86_64
- Docker-compose: 1.2.0
- Apache Maven: 3.2.5
- MRC (*Matlab Component Runtime*): 9.0, descargable desde la web http://es.mathworks.com/products/compiler/mcr/.

Cabe destacar que: si no se usa la versión 22 de Fedora como sistema operativo Linux, habrá que ajustar las versiones de los servicios utilizados a la nueva versión; exceptuando la versión del MCR, que habrá que actualizarla si se vuelve a compilar la rutina de Matlab con una versión de Matlab distinta a la 2015b, utilizada en este proyecto.

Una vez instalados todos los servicios necesarios, ya sólo queda ejecutarlos junto a todas las aplicaciones (con permisos de administrador del sistema):

- Para encender el servicio de base de datos, si no se inicia por defecto, hay que ejecutar la sentencia siguiente: *mysqld_safe &*
- Para ejecutar el servicio Docker basta con escribir, también en la consola: *docker daemon &*
- Ahora, hay que ir al directorio raíz de la carpeta con nombre “java”, donde se encuentran todas las aplicaciones y el fichero *docker-compose.yml*; el cual tiene la configuración de arranque del servicio Redis y ejecutar el comando *docker-compose up*.
- Primero, hay que instalar la base de datos: Crear la base de datos structures y ejecutar el comando: *mysql –u root < structures.sql*, donde se puede ver que, por el momento, no hay password para el usuario *root* (situación de testeo).
- El *Matlab Component Runtime* descargado hay que situarlo en el directorio /usr/local/MATLAB/MATLAB_Runtime/v90/
- Y, por último, hay que ir al directorio raíz de cada aplicación, donde se encuentra el fichero pom.xml y ejecutar el comando de maven: *mvn spring-boot:run*. Las aplicaciones son: *gateway*, *structures*, *uiBasicUser, uiAdminUser* y *uiSuperAdminUser*.

8. Conclusiones

Para abordar este capítulo me voy a remitir al primero, comentaré algunos de las ventajas e inconvenientes detectados que supone la realización del proyecto y propondré algunos aspectos para la posible ampliación del mismo.

En primer lugar, tal como especifican los objetivos, en el apartado 1.2 del capítulo primero, el proyecto se ha basado en crear una aplicación de entorno Web, donde su finalidad principal fuese la de permitir calcular estructuras de sección rectangular, con ayuda de una rutina específica para ello, realizada en entorno Matlab. Además, a parte del objetivo principal, se han realizado dos interfaces gráficas más; una para la gestión de usuarios y otra para las variables fijas de entrada al cálculo de la estructura.

Este sistema se ha realizado tal como se ha descrito a lo largo de los siete capítulos anteriores, intentando separar conceptos con el fin de facilitar los posibles futuros desarrollos y ampliaciones del mismo.

Ventajas

Algunas de las ventajas de realizar un servicio web de estas características son las siguientes:

- El servicio se puede ofrecer a nivel mundial, por el simple hecho de estar disponible en internet.
- La interfaz gráfica se adapta a múltiples dispositivos; dado que la aplicación es "*responsive*" se adapta a todos los dispositivos que hagan uso del navegador para acceder a internet.
- Por el hecho de tener una API de consulta a la base de datos y la aplicación de Matlab, permite el acceso a los datos por parte de otros programas y/o aplicaciones móviles.
- Una de las mayores ventajas es que, gracias al Matlab *Component Runtime*, no es necesario ninguna licencia.
- Dada la estructura que se ha propuesto de micro-servicios, permite aumentar el número de servicios independientes de los ya creados. Incluso permite crear grupos específicos de programadores por micro-servicio.
- No es necesaria la intervención de la interface gráfica de Matlab, así como tampoco es necesario tener conocimientos ni incluso saber que se basa en Matlab, para usarla.

Inconvenientes

Aun así, también se han detectado algunos inconvenientes como los que se describen a continuación:

- El hecho de no necesitar la interface gráfica de Matlab ya se ha detectado como una ventaja, aun así, también se convierte en un inconveniente respecto a la pérdida de su riqueza y, en caso de querer tenerla, habría que generarla para el entorno Web.
- El hecho de haber creado una aplicación por cada servicio, dificulta la estructura de la misma, aumentando, por ejemplo, la comunicación entre equipos de programadores.
- La estructura de la base de datos, también, aumentará su grado de dificultad cuando las aplicaciones sean totalmente distribuidas.

Ampliaciones y mejoras

En lo que a las ampliaciones futuras se refiere, podría verse el sistema como un servicio de cálculos; donde cada tipo de cálculo se albergara en un micro-servicio distinto, con su propia rutina de Matlab, generando así una especie de calculadora específica para dar solución a tipos de problemas estándar.

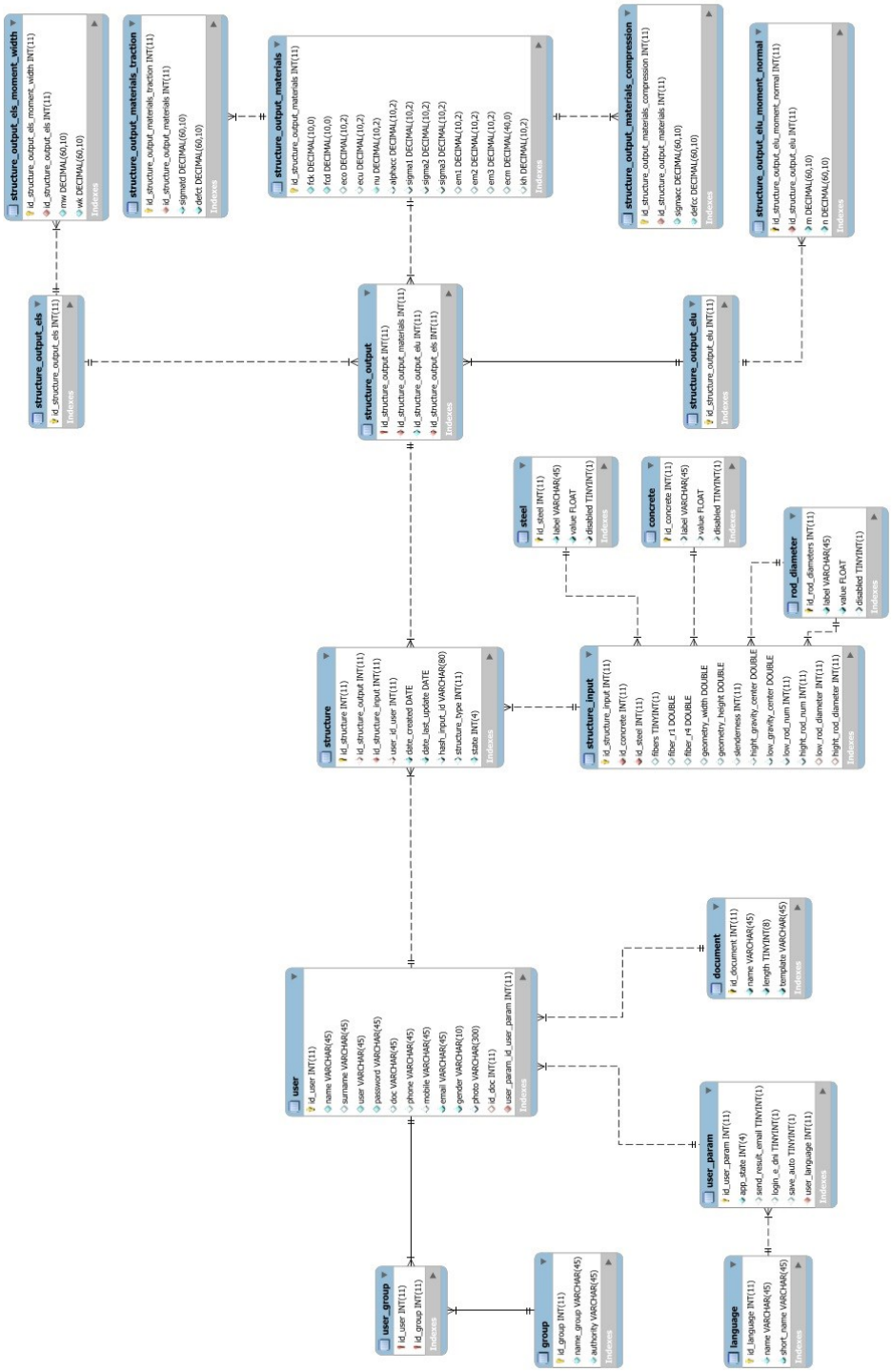
Hay varias mejoras que se podrían hacer, como son:

- Estudiar el alcance de los gráficos vectoriales redimensionables o **SVG** (del inglés, Scalable Vector Graphics) para enriquecer la representación de gráficas.
- Como se ha comentado en el capítulo de requisitos; se podría implementar, a partir del protocolo OAUTH2, un servicio de autenticación de usuarios con servidores de identidades como Google y Facebook, entre otros.

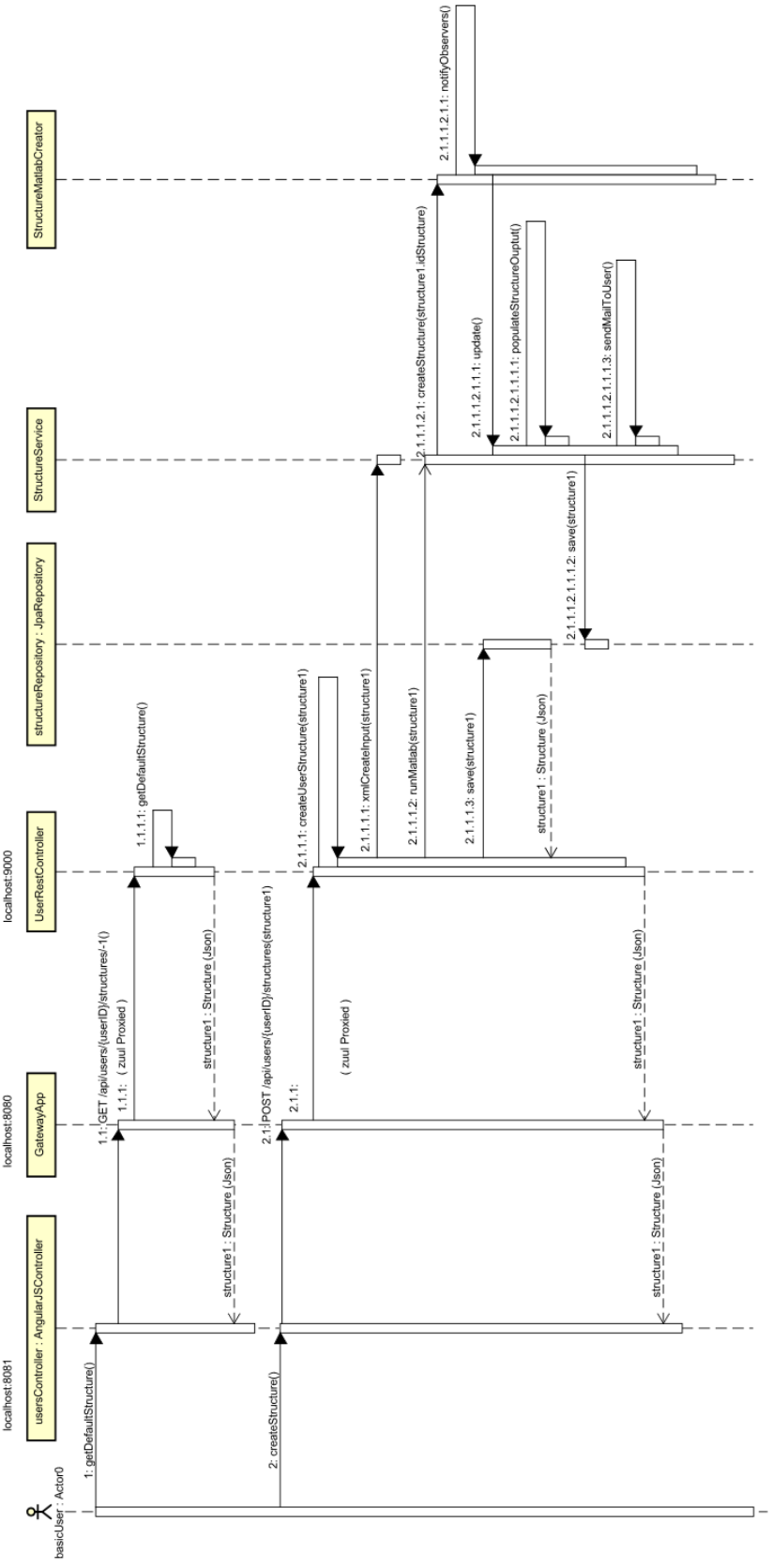
Finalmente, respecto a la finalización del proyecto, se debe decir que, como toda aplicación web, hay muchos aspectos tanto de diseño como de implementación mejorables y, por supuesto, no se han podido abarcar todos.

9. Apéndice

9.1 Base de datos completa



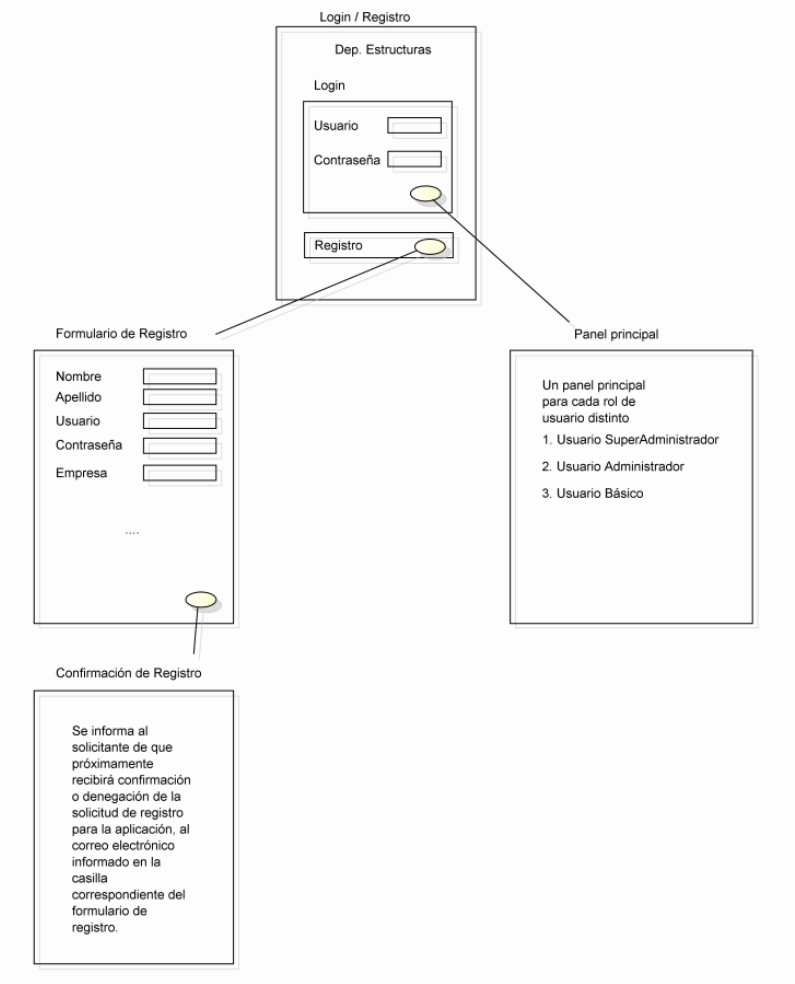
9.2 DS: Calcular estructura



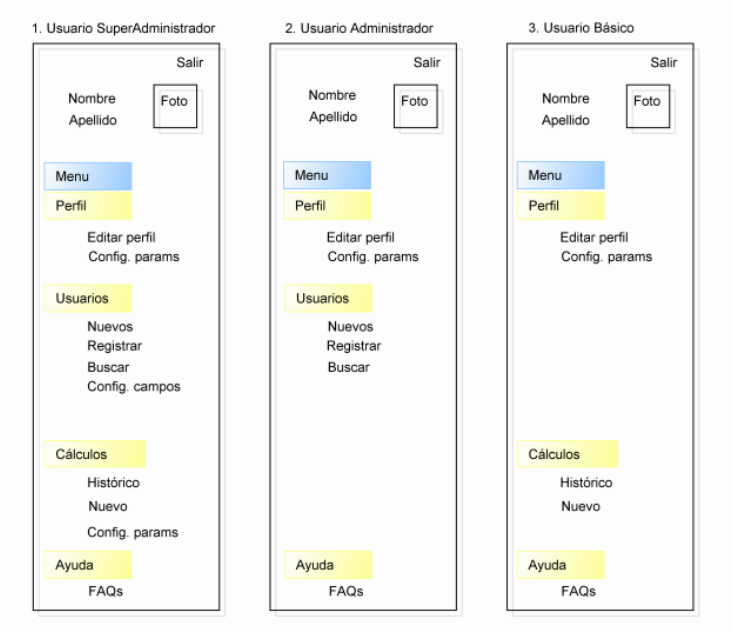
9.3 Diseño inicial

Con el fin de agilizar la comunicación, antes de empezar a realizar la versión final del proyecto, en a principios de Enero del 2015, se realizaron unos bocetos iniciales de la interface gráfica web; los cuales se muestran a continuación.

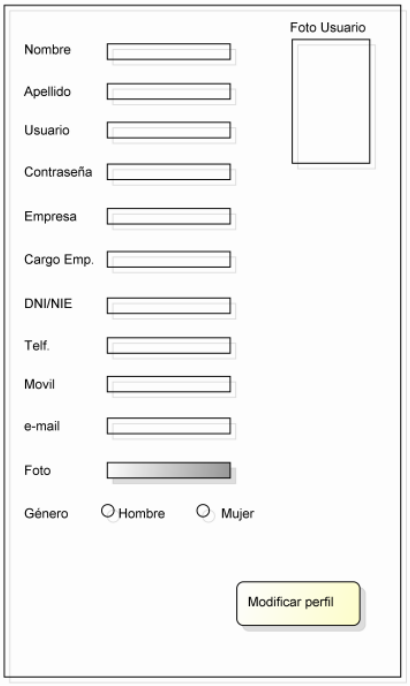
Página inicial de login y solicitud de registro



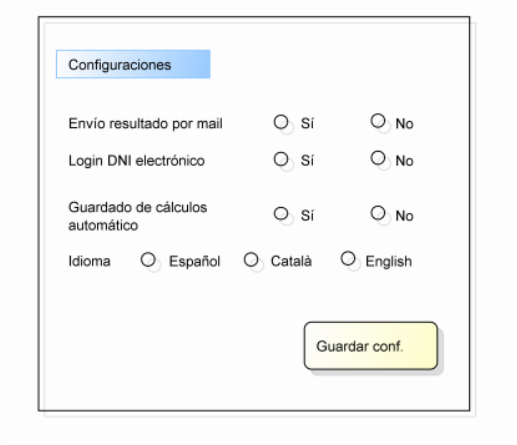
Menús de usuario por perfil



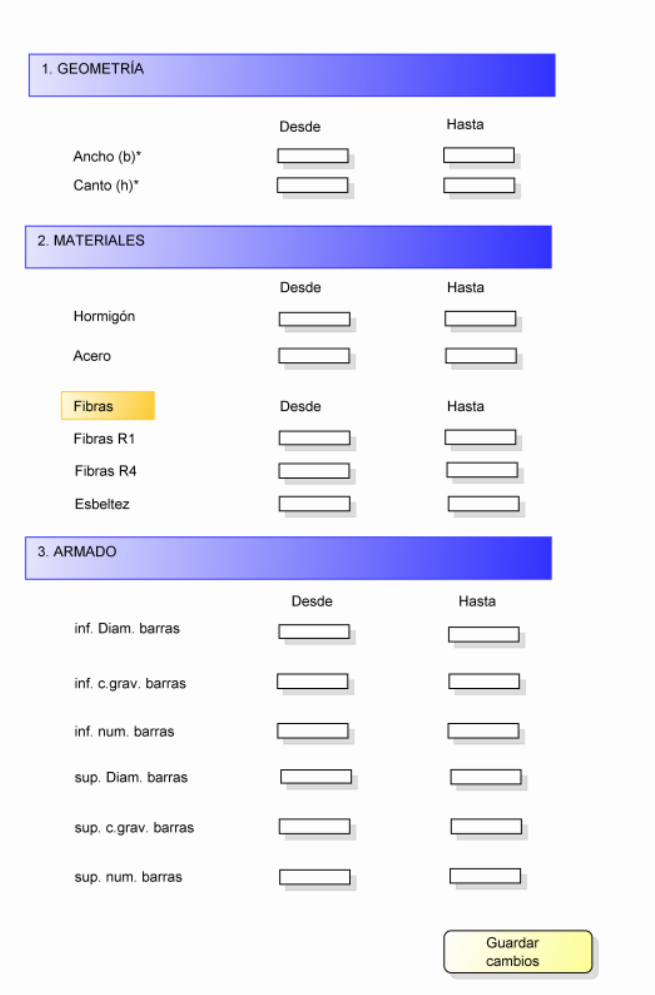
Perfil de usuario



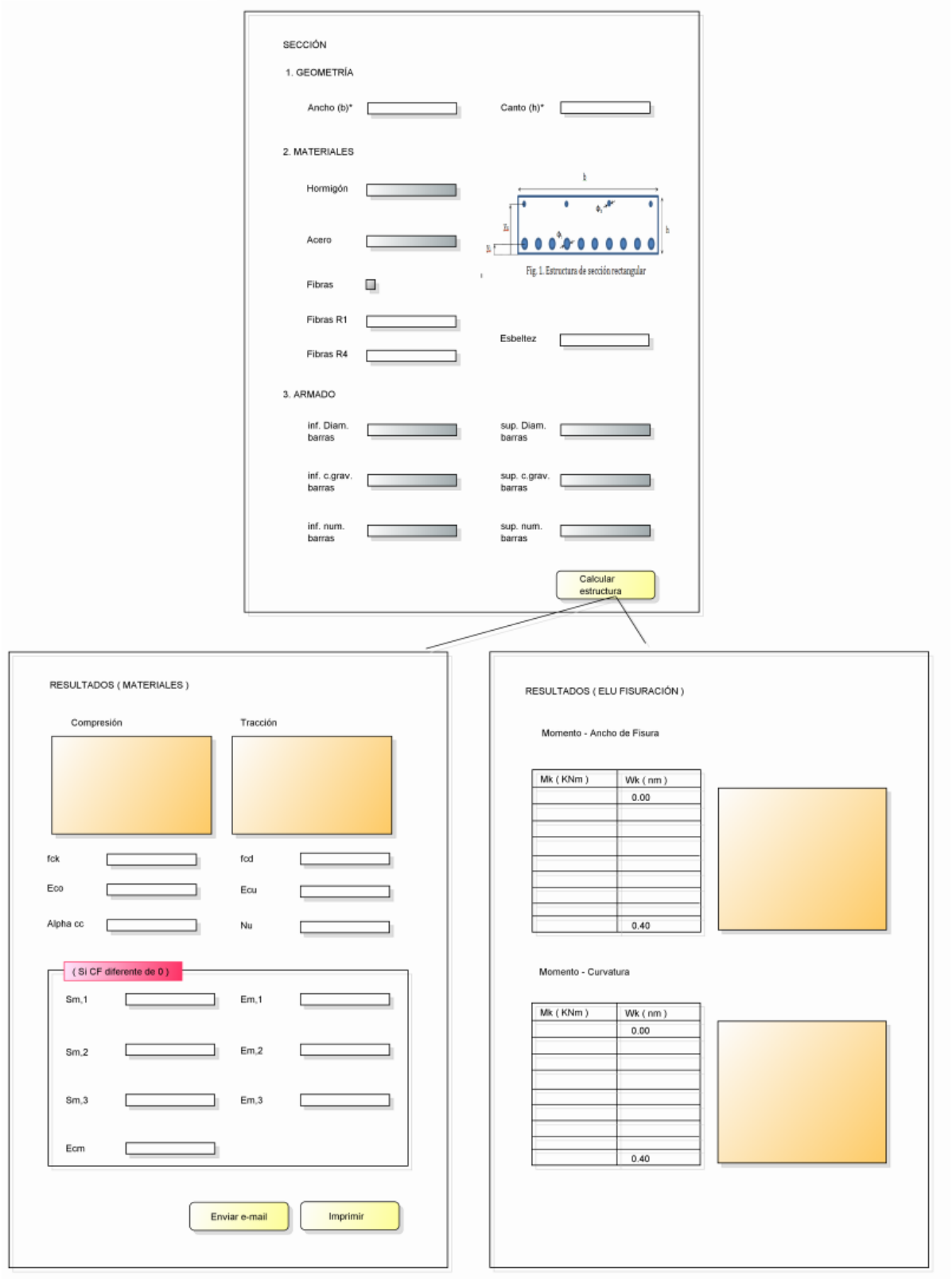
Parámetros de configuración de usuario



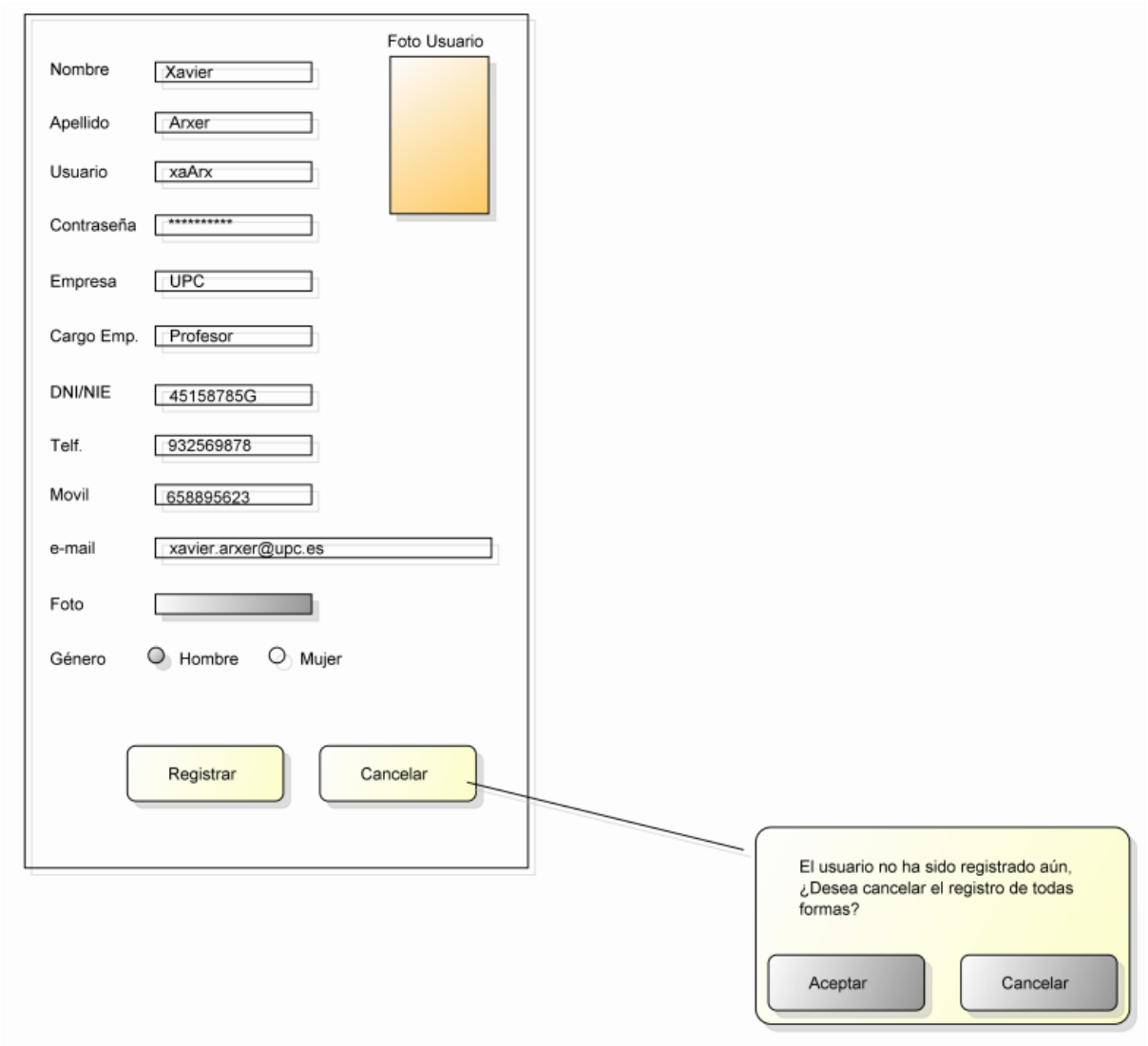
Parámetros de configuración del súper usuario



Cálculo de la estructura



Registro de usuario por parte del administrador



10. Referencias

Protocolo HTTP:

- http://es.ccm.net/contents/264-el-protocolo-http
- https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- https://es.wikipedia.org/wiki/Request_for_Comments
- https://es.wikipedia.org/wiki/Grupo_de_trabajo_de_ingenier%C3%ADa_de_internet
- https://tools.ietf.org/html/rfc2616
- https://es.wikipedia.org/wiki/HTTP/2
- https://tools.ietf.org/html/rfc2145

HTML, CSS, Bootstrap y Angular JS

- http://www.codeproject.com/Articles/567385/CSSplusBoxplusModelplusandplusPositioning
- https://es.wikipedia.org/wiki/HTML
- https://es.wikipedia.org/wiki/Etiqueta_meta
- https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada
- http://www.w3schools.com/css/
- http://www.w3schools.com/css/default.asp
- http://www.w3schools.com/css/css3_intro.asp
- http://www.w3schools.com/css/css_boxmodel.asp
- http://www.w3schools.com/bootstrap/bootstrap_ref_js_tab.asp
- http://getbootstrap.com/css/
- https://angular-ui.github.io/bootstrap/
- https://docs.angularjs.org/api/
- https://es.wikipedia.org/wiki/AngularJS
- https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Trabajando_con_objectos
- http://www.w3schools.com/angular/default.asp
- http://www.w3schools.com/charsets/ref_utf_greek.asp
- http://www.w3schools.com/html/default.asp
- http://www.w3schools.com/html/html5_intro.asp
- http://www.w3schools.com/js/default.asp

Api REST

- https://es.wikipedia.org/wiki/Representational_State_Transfer
- http://www.restapitutorial.com/

Java y POO

- https://en.wikipedia.org/wiki/Object-oriented_programming
- http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=419:tipos-de-datos-java-tipos-primitivos-int-boolean-y-objeto-string-array-o-

arreglo-variables-cu00621b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188
- http://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html#jls-3.10.2
- https://es.wikipedia.org/wiki/Tipado_fuerte

Spring framework
- http://www.it-ebooks.info/book/4983/
- http://start.spring.io/
- https://spring.io/blog/2015/03/18/spring-boot-support-in-spring-tool-suite-3-6-4
- http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-servlet
- http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-introduction
- http://docs.spring.io/spring-data/jpa/docs/1.10.2.RELEASE/reference/html/#repositories
- http://spring.io/guides
- http://spring.io/guides/gs/rest-service/
- http://spring.io/guides/gs/consuming-rest-angularjs/
- http://spring.io/guides/tutorials/bookmarks/
- http://www.tuprogramacion.com/glosario/que-es-un-orm/
- https://es.wikipedia.org/wiki/Mapeo_objeto-relacional
- http://hibernate.org/orm/documentation/5.0/
- http://maven.apache.org/
- http://maven.apache.org/pom.html
- https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html
- http://www.mkyong.com/spring/maven-spring-hibernate-mysql-example/
- https://spring.io/guides/tutorials/spring-security-and-angular-js/

SQL, MySQL y MariaDB
- https://dev.mysql.com/
- http://www.w3schools.com/sql/default.asp
- https://es.wikipedia.org/wiki/SQL#cite_note-1
- https://es.wikipedia.org/wiki/MySQL
- https://es.wikipedia.org/wiki/MariaDB

Matlab y MCR
- http://es.mathworks.com/help/compiler/index.html
- http://es.mathworks.com/help/compiler/deployment-process.html
- http://es.mathworks.com/help/compiler/standalone-applications.html?requestedDomain=es.mathworks.com
- http://es.mathworks.com/products/compiler/mcr/
- http://es.mathworks.com/products/compiler/features.html#uso-compartido-de-programas-de-matlab-como-aplicaciones-aut%C3%B3nomas

- https://es.wikipedia.org/wiki/MATLAB

MVVM y SPA

- https://es.wikipedia.org/wiki/Patrón_de_diseño
- https://en.wikipedia.org/wiki/Model-view-viewmodel
- https://es.wikipedia.org/wiki/Single-page_application
- https://es.wikipedia.org/wiki/Observer_(patrón_de_diseño)

Gateway y Microservicios

- https://tools.ietf.org/html/rfc2616
- http://microservices.io/patterns/microservices.html
- http://microservices.io/articles/scalecube.html
- http://microservices.io/patterns/apigateway.html

Diagramas de secuencia

- http://www.ibm.com/developerworks/rational/library/3101.html
- http://www.agilemodeling.com/artifacts/sequenceDiagram.htm

Otros

- http://stackoverflow.com/questions

Ilustraciones Externas:

- Ilustración 1-1:
  Proporcionada por el departamento de Ingeniería de la Construcción de la Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos de Barcelona.

- Ilustración 2-1:
  https://blogs.technet.microsoft.com/netmon/2007/12/21/understanding-http-flow-with-netmon-3-by-yuri-diogenes/

- Ilustración 2-2:
  https://es.wikipedia.org/wiki/HTML

- Ilustración 2-3:
  http://www.codeproject.com/Articles/567385/CSSplusBoxplusModelplusandplusPositioning

- Ilustración 2-4:
  http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-basics

- Ilustración 2-5:
  http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-servlet

- Ilustración 5-1:
  https://en.wikipedia.org/wiki/Model-view-viewmodel

SOFTWARE:

Astah community (Diagramas)

- http://astah.net/download/community?http%3A//cdn.astah.net/downloads/astah-community-7_0_0-846701-jre-64bit-setup.exe

Astah profesional (Diagramas) (con licencia para estudiantes)

- http://astah.net/download

MySQL Workbench

- https://dev.mysql.com/downloads/file/?id=460621

Matlab y MCR

- El Matlab, y su correspondiente versión del MCR, han sido aportados por el “Servei de distribució de Software” de la UPC:
  https://distribuciosoftware.upc.edu